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
2.2 OK . . . 21
2.2.1 <break> dans un <for>, apres instruction . . . 21
2.2.2 <break> dans un <for>, sans instruction avant . . . 23
2.2.3 fonction executant seulement <break> . . . 25
2.2.4 appel de <break>, avec corps de programme ne contenant pas <break> . . . 27
2.2.5 <break> dans un <while>, apres instruction . . . 29
2.2.6 <break> dans un <while>, sans instruction avant . . . 31
2.2.7 <break> dans un <if then>, apres instruction . . . 33
2.2.8 <break> dans un <if then>, sans instruction avant . . . 35
2.2.9 <break> dans un <if then else>, apres instruction . . . 37
2.2.10 <break> dans un <if then else>, sans instruction avant . . . 39
2.2.11 <break> seul . . . 41
2.2.12 <break> seul, 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
3.2 OK . . . 79
3.2.1 addition simple, a 2 termes . . . 79
3.2.2 soustraction simple, a 2 termes . . . 81
3.2.3 multiplication simple, a 2 termes . . . 83
3.2.4 division simple, a 2 termes . . . 85
3.2.5 addition a 3 termes . . . 87

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

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

3.2.102 3 instructions sans parentheses . . . 281
3.2.103 concatenation d'entier et de chaine . . . 283
3.2.104 concatenation de 2 chaines . . . 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 incrementation dans la condition <if then> . . . 497
9.1.2 affectation d'entier dans la condition <if then> . . . 499
9.1.3 affectation de chaine dans la condition <if then> . . . 501
9.1.4 <if then> avec oubli du <if> . . . 503
9.1.5 <if then> avec oubli du <then> . . . 505
9.1.6 <if then> avec oubli de la condition . . . 507
9.1.7 <if then else> avec oubli de la condition . . . 509
9.1.8 <if then else> sans instruction dans <else> . . . 511
9.1.9 <if then else> avec ligne vide dans <else> . . . 513
9.1.10 <if then> avec condition entiere . . . 515
9.2 OK . . . 517
9.2.1 <if then> sans instruction . . . 517
9.2.2 <if then> avec ligne vide . . . 519
9.2.3 <if then> avec simple condition . . . 521
9.2.4 <if then> avec double-condition . . . 523
9.2.5 <if then> avec triple-condition . . . 525
9.2.6 double-imbrication de <if then> . . . 527
9.2.7 triple-imbrication de <if then> . . . 529
9.2.8 double-imbrication de <if then else> . . . 531
9.2.9 triple-imbrication de <if then else> . . . 533

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 var **577**
12.1 KO . . . 577
12.1.1 declaration sans affectation . . . 577
12.1.2 declaration d'entier avec type sans affectation . . . 579
12.1.3 declaration de chaine avec type sans affectation . . . 581
12.1.4 affectation d'entier sans declaration . . . 583
12.1.5 affectation de chaine sans declaration . . . 585
12.1.6 affectation de <nil> . . . 587
12.1.7 double-declaration avec valeurs egales . . . 589
12.1.8 double-declaration avec valeurs differentes . . . 591
12.1.9 <var> mal ecrit . . . 593
12.1.10 declaration de caractere special . . . 595
12.1.11 affectation de caractere special . . . 597
12.1.12 affectation d'operation logique sur entiers . . . 599
12.1.13 affectation d'operation logique sur chaines . . . 601
12.1.14 chiffre comme nom de variable . . . 603
12.1.15 <_> comme nom de variable . . . 605
12.2 OK . . . 607
12.2.1 declaration d'entier simple . . . 607
12.2.2 declarations d'entier avec reutilisation de 2 autres entiers . . . 609
12.2.3 declarations d'entier avec reutilisation de 3 autres entiers . . . 611
12.2.4 declaration simple d'entier avec type . . . 613
12.2.5 declarations d'entier avec types et reutilisation de 2 autres entiers . . . 615
12.2.6 declarations d'entier avec types et reutilisation de 3 autres entiers . . . 617
12.2.7 declaration de chaine simple . . . 619
12.2.8 declarations de chaine avec reutilisation de 2 autres chaines . . . 621
12.2.9 declarations de chaine avec reutilisation de 3 autres chaines . . . 623
12.2.10 declaration de chaine avec type . . . 625
12.2.11 declarations de chaine avec types et reutilisation de 2 autres chaines . . . 627
12.2.12 declarations de chaine avec types et reutilisation de 3 autres chaines . . . 629
12.2.13 lettre minuscule comme nom de variable . . . 631
12.2.14 lettre majuscule comme nom de variable . . . 633
12.2.15 lettre minuscule et chiffre dans nom de variable . . . 635
12.2.16 lettre majuscule et chiffre dans nom de variable . . . 637
12.2.17 lettre minuscule et <_> dans nom de variable . . . 639
12.2.18 lettre majuscule et <_> dans nom de variable . . . 641
12.2.19 lettre minuscule, chiffre et <_> dans nom de variable . . . 643
12.2.20 lettre majuscule, chiffre et <_> dans nom de variable . . . 645

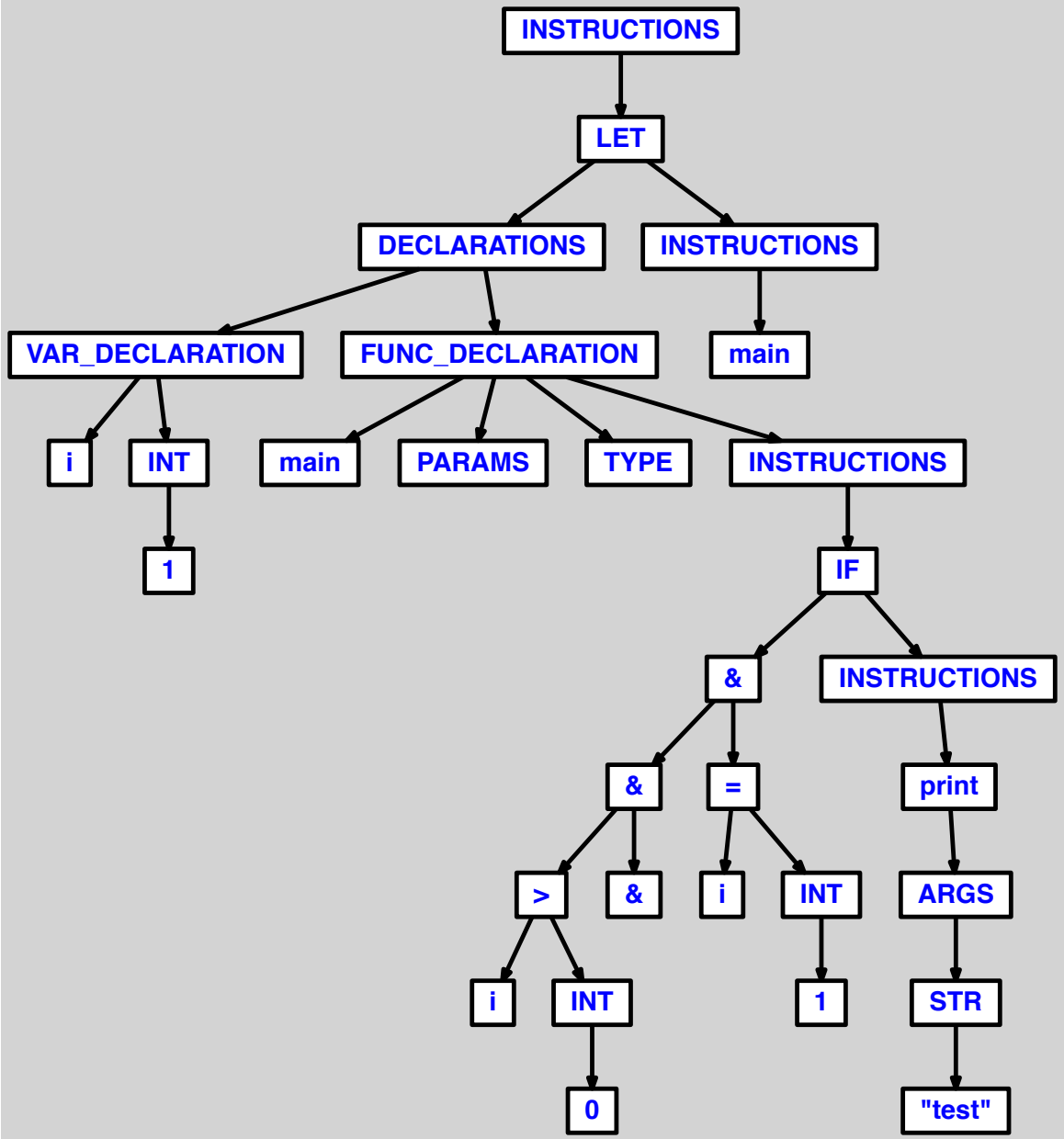
13 while **647**
13.1 KO . . . 647
13.1.1 <while> avec affectation d'entier . . . 647
13.1.2 <while> avec affectation de chaine . . . 649
13.1.3 <while> avec oubli du compteur . . . 651
13.1.4 <while> avec oubli du <while> . . . 653
13.1.5 <while> avec oubli du <do> . . . 655
13.1.6 <while> avec oubli de la condition . . . 657
13.2 OK . . . 659
13.2.1 <while> avec iteration croissante . . . 659
13.2.2 <while> avec iteration decroissante . . . 661
13.2.3 <while> avec double-condition . . . 663
13.2.4 <while> avec triple-condition . . . 665
13.2.5 <while> sans instruction . . . 667
13.2.6 <while> avec ligne vide . . . 669
13.2.7 <while> avec condition entiere . . . 671
13.2.8 <while> avec double-imbrication . . . 673
13.2.9 <while> avec triple-imbrication . . . 675
13.2.10 <while> avec reutilisation de compteur . . . 677
13.2.11 <while> avec reutilisation de compteurs . . . 679
13.2.12 <while> avec iteration sur une chaine . . . 681

1 and

1.1 KO

1.1.1 <&> mal ecrit

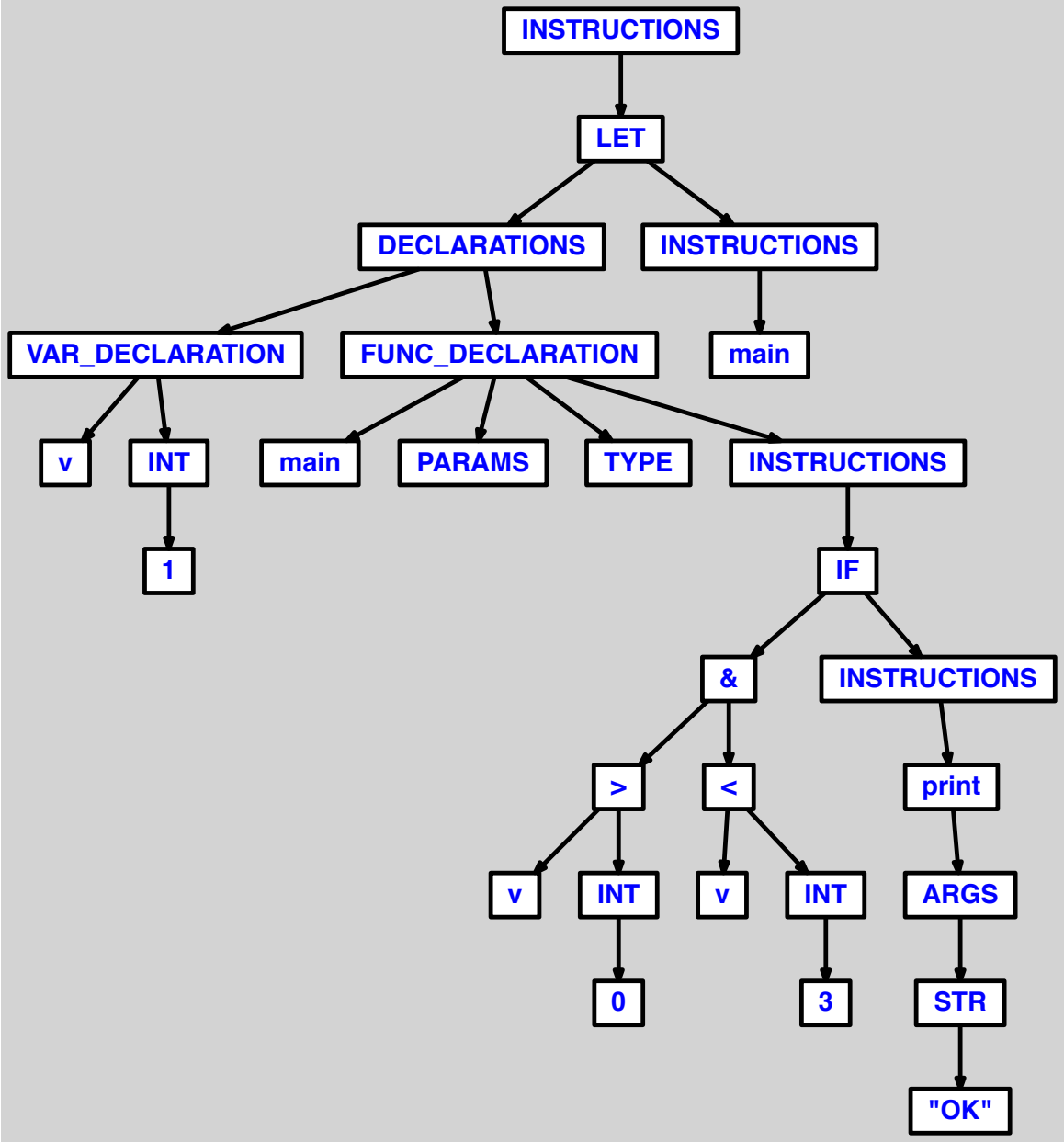
```
let
  var i := 1

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


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

1.2 OK

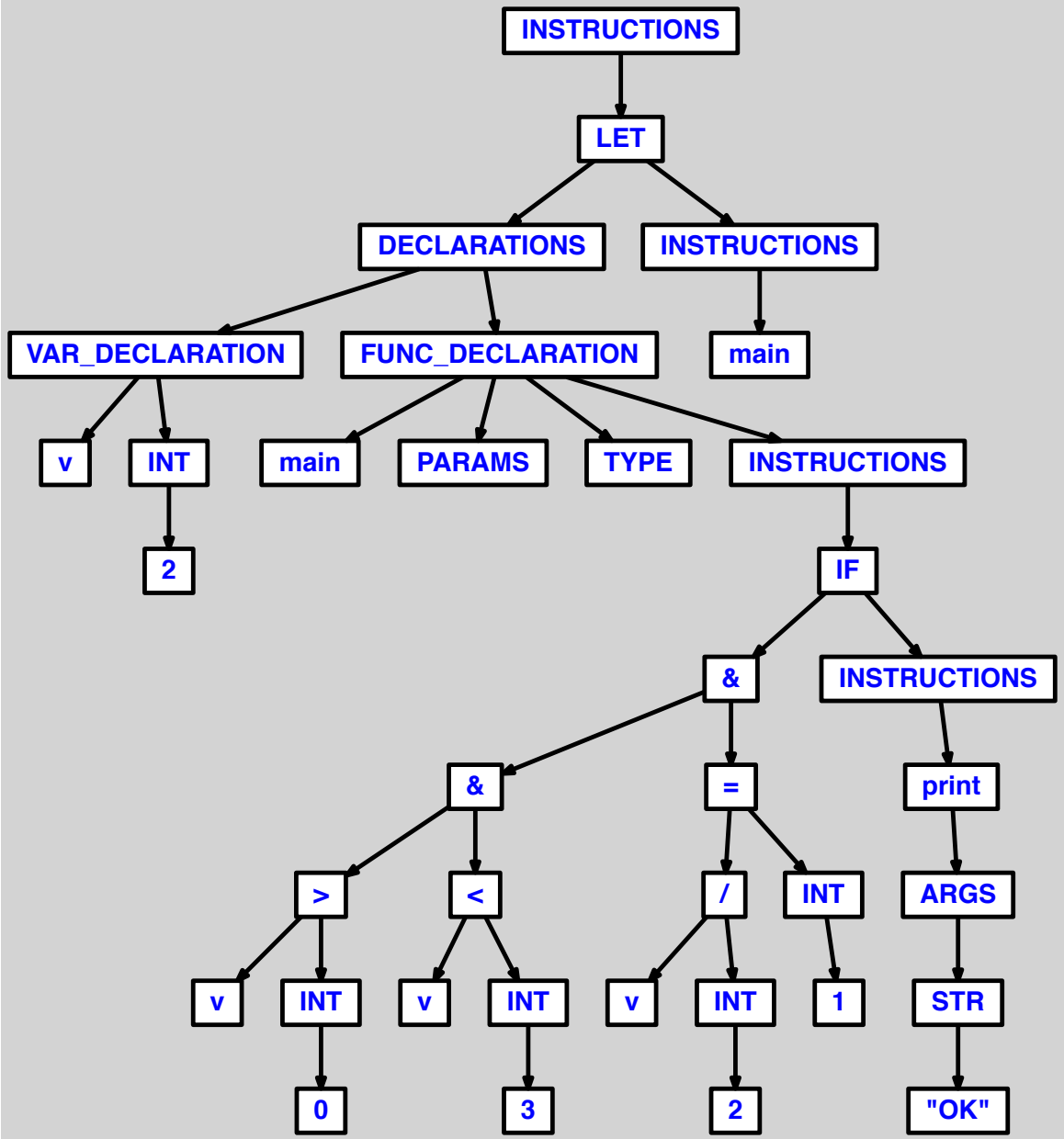
1.2.1 condition avec 1 <&>

```
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
main
v
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
IF
&
INSTRUCTIONS
>
<
print
v
INT
v
INT
ARGS
0
3
STR
"OK"

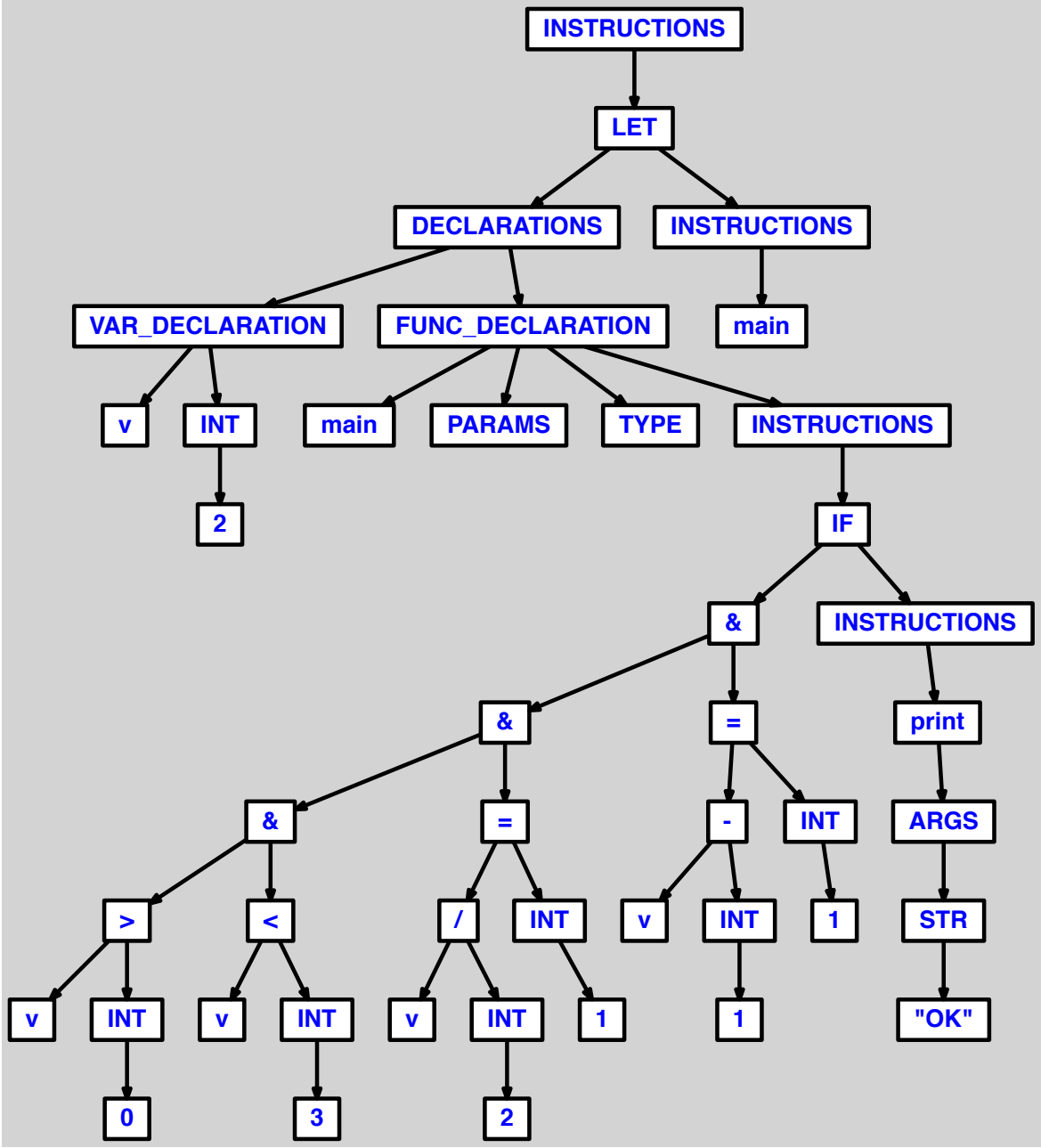
1.2.2 condition avec 2 <&>

```
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
main
v
INT
main
PARAMS
TYPE
INSTRUCTIONS
2
IF
&
INSTRUCTIONS
&
=
print
>
<
/
INT
ARGS
v
INT
v
INT
v
INT
1
STR
0
3
2
"OK"

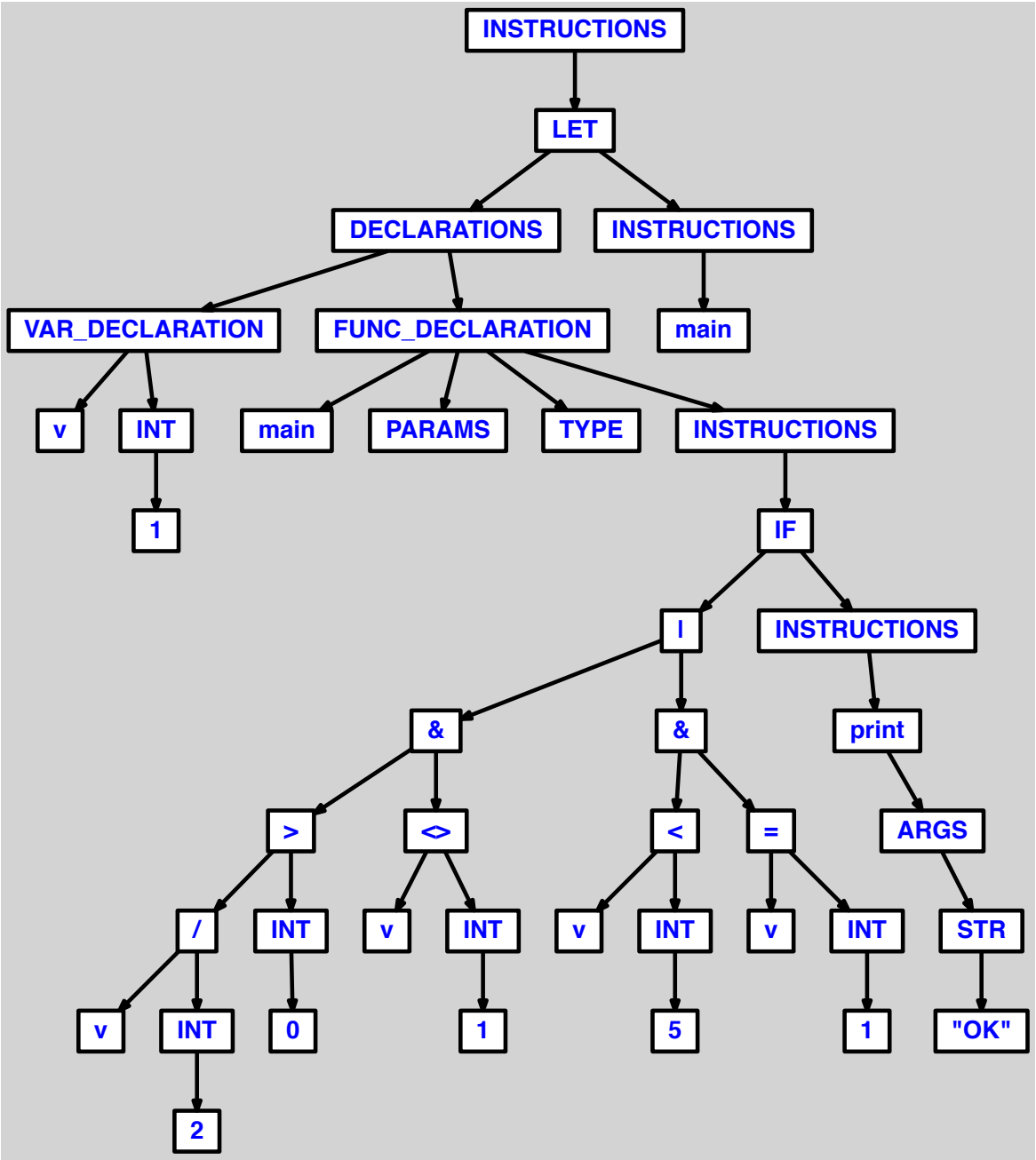
1.2.3 condition avec 3 <&>

```
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
main
v
INT
main
PARAMS
TYPE
INSTRUCTIONS
2
IF
&
INSTRUCTIONS
&
=
print
&
=
-
INT
ARGS
>
<
/
INT
v
INT
1
STR
v
INT
v
INT
v
INT
1
1
"OK"
0
3
2

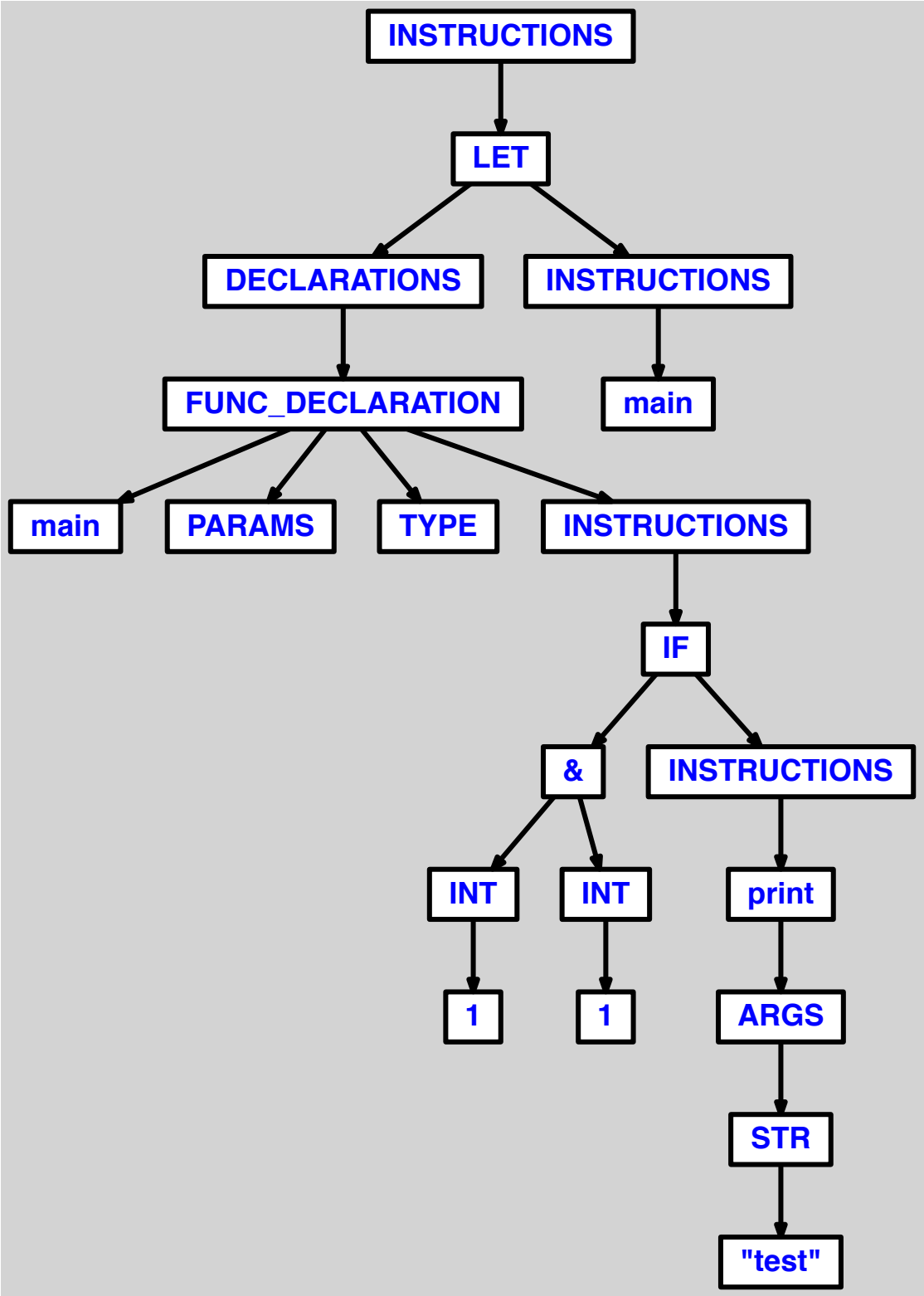
1.2.4 condition avec 2 <&> et 1 <|>

```
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
main
v
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
IF
|
INSTRUCTIONS
&
&
print
>
<>
<
=
ARGS
/
INT
v
INT
v
INT
v
INT
STR
v
INT
0
1
5
1
"OK"
2

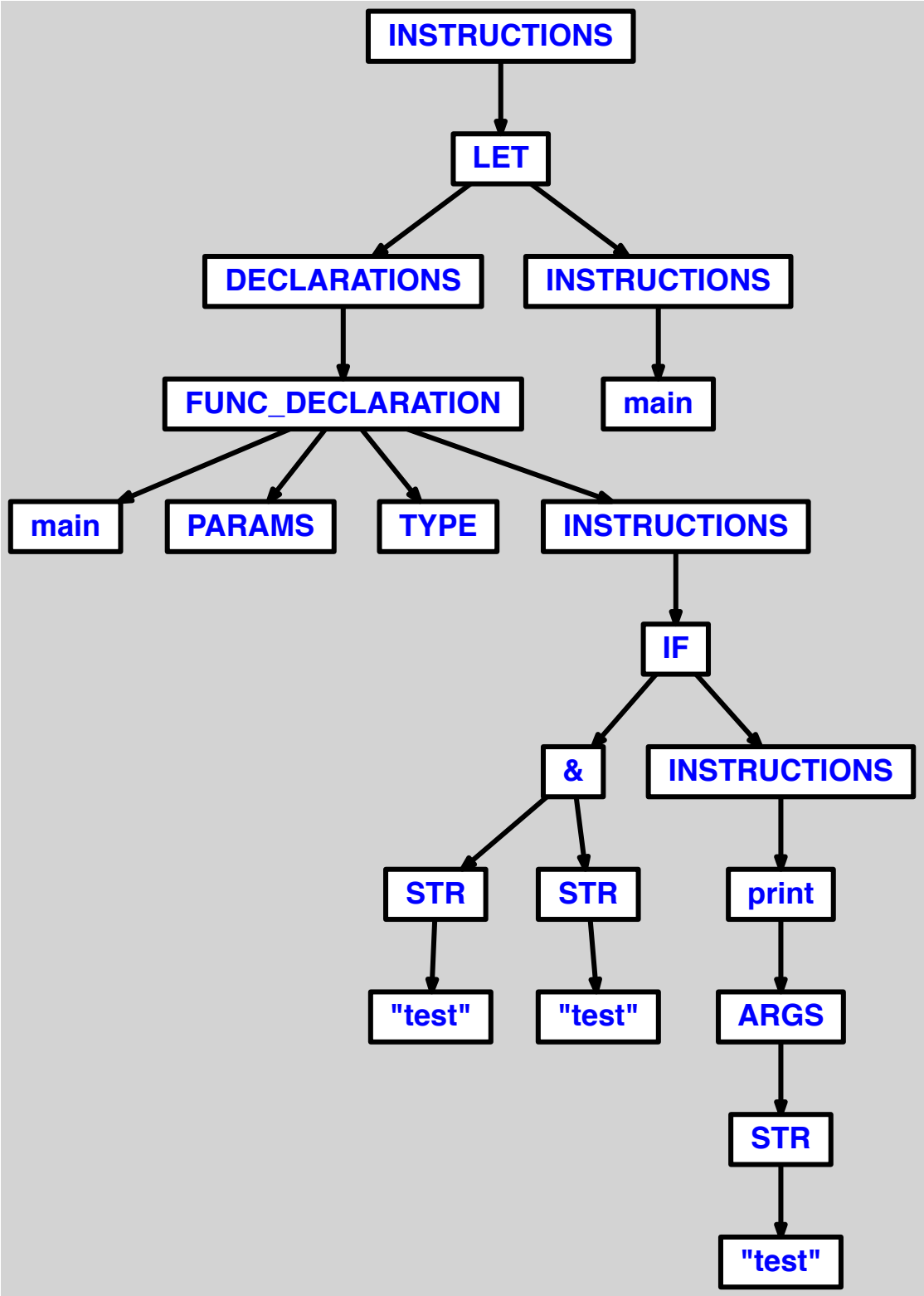
1.2.5 <&> avec des entiers

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


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

1.2.6 <&> avec des chaines

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

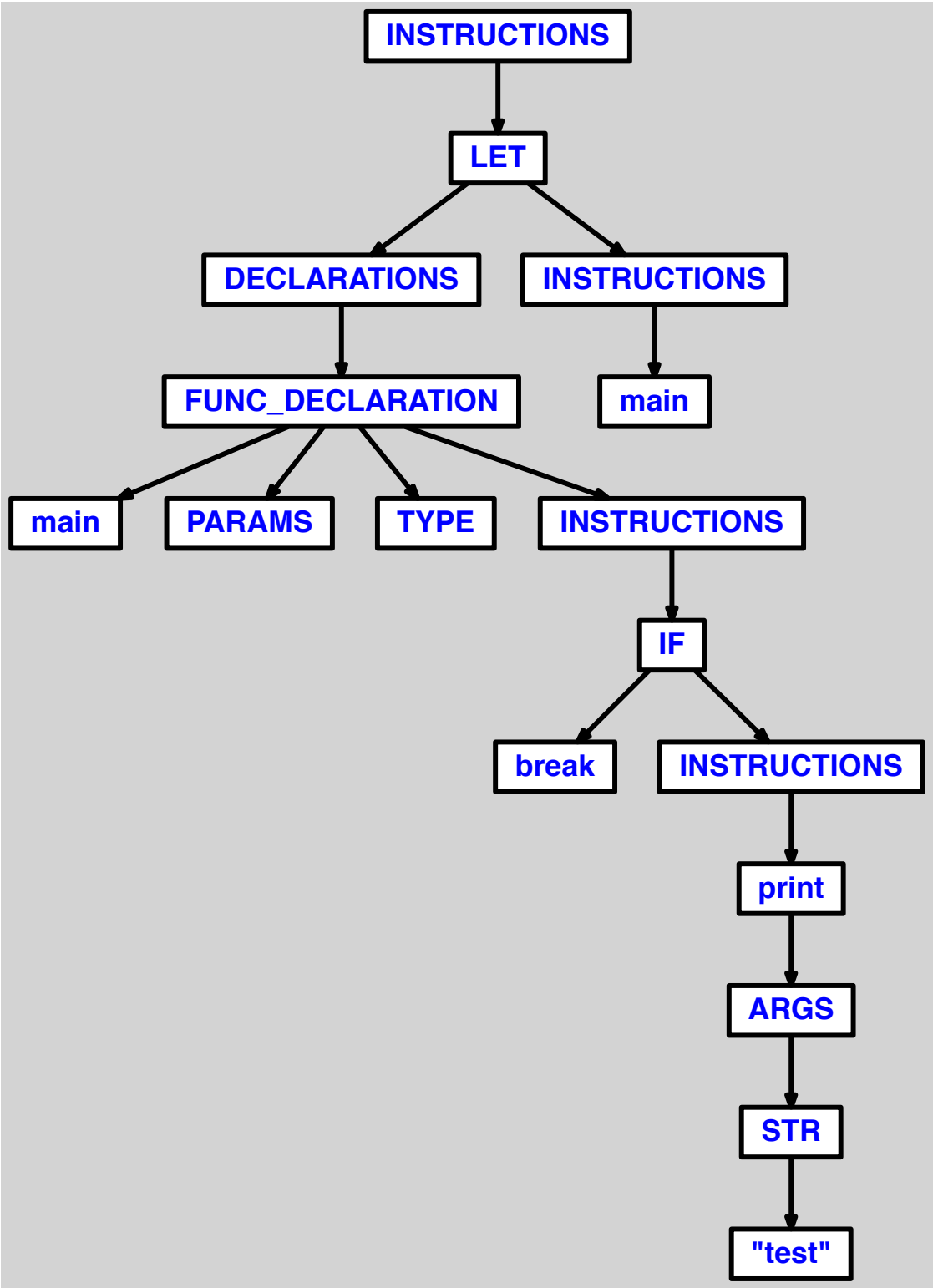

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
IF
&
INSTRUCTIONS
STR
STR
print
"test"
"test"
ARGS
STR
"test"

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
main
main
PARAMS
TYPE
INSTRUCTIONS
IF
break
INSTRUCTIONS
print
ARGS
STR
"test"

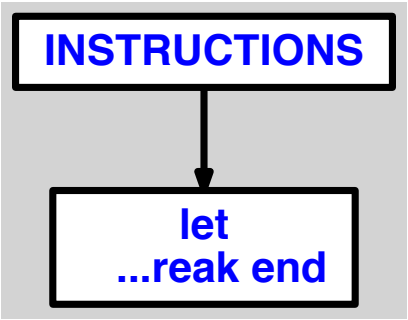
2.1.2 <break> comme condition dans <while>

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

Pas d'AST, problème de syntaxe.

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

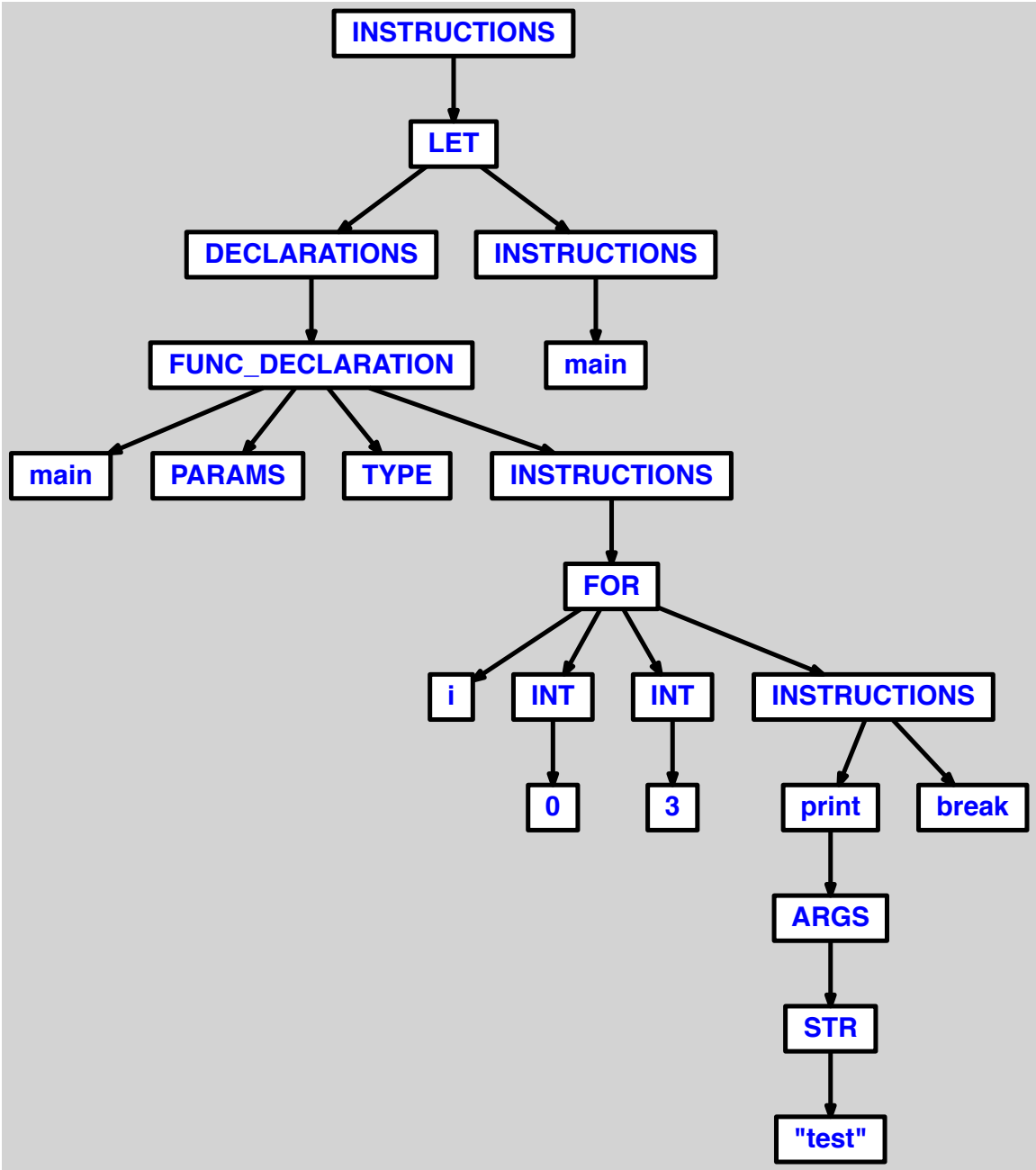
```
let
  break
in break end
```


INSTRUCTIONS
let
...reak end

2.2 OK

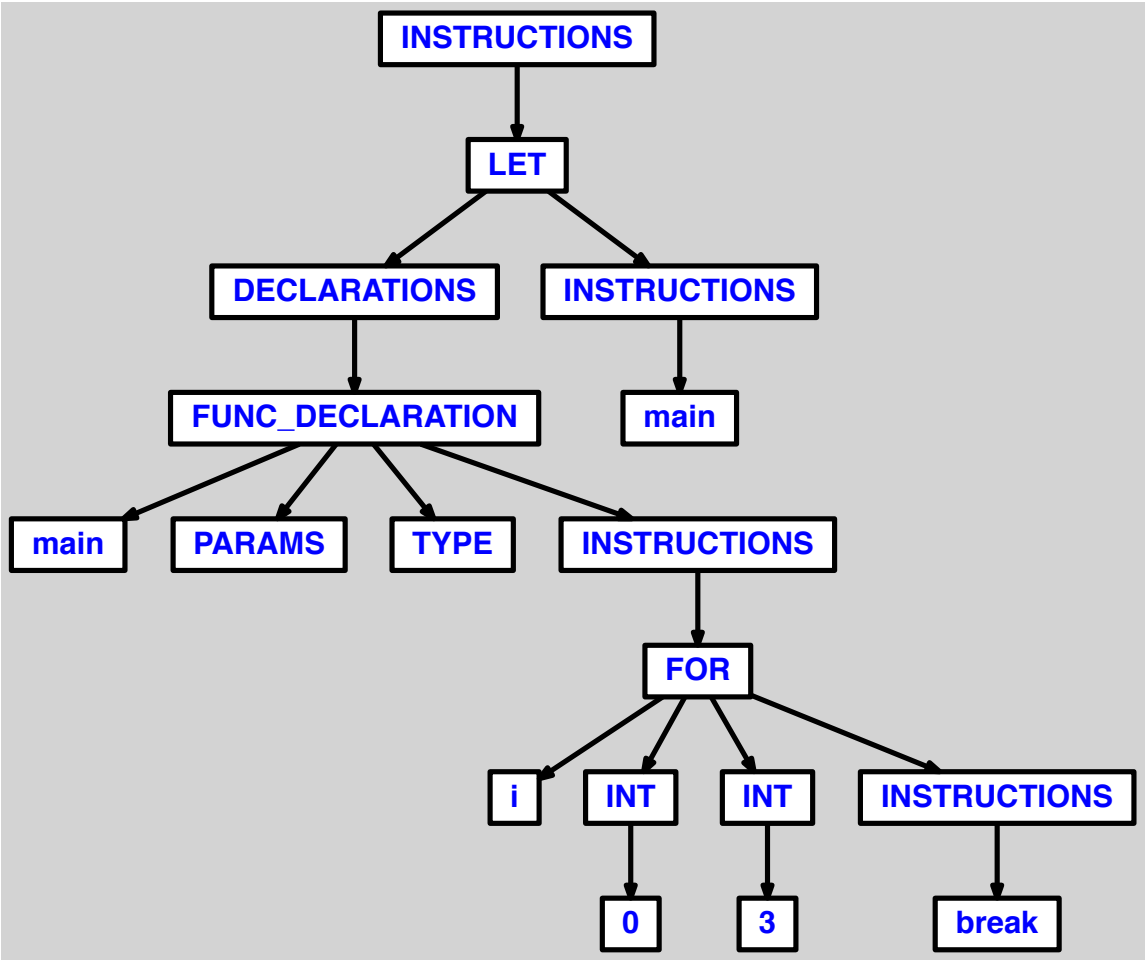
2.2.1 <break> dans un <for>, apres instruction

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


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

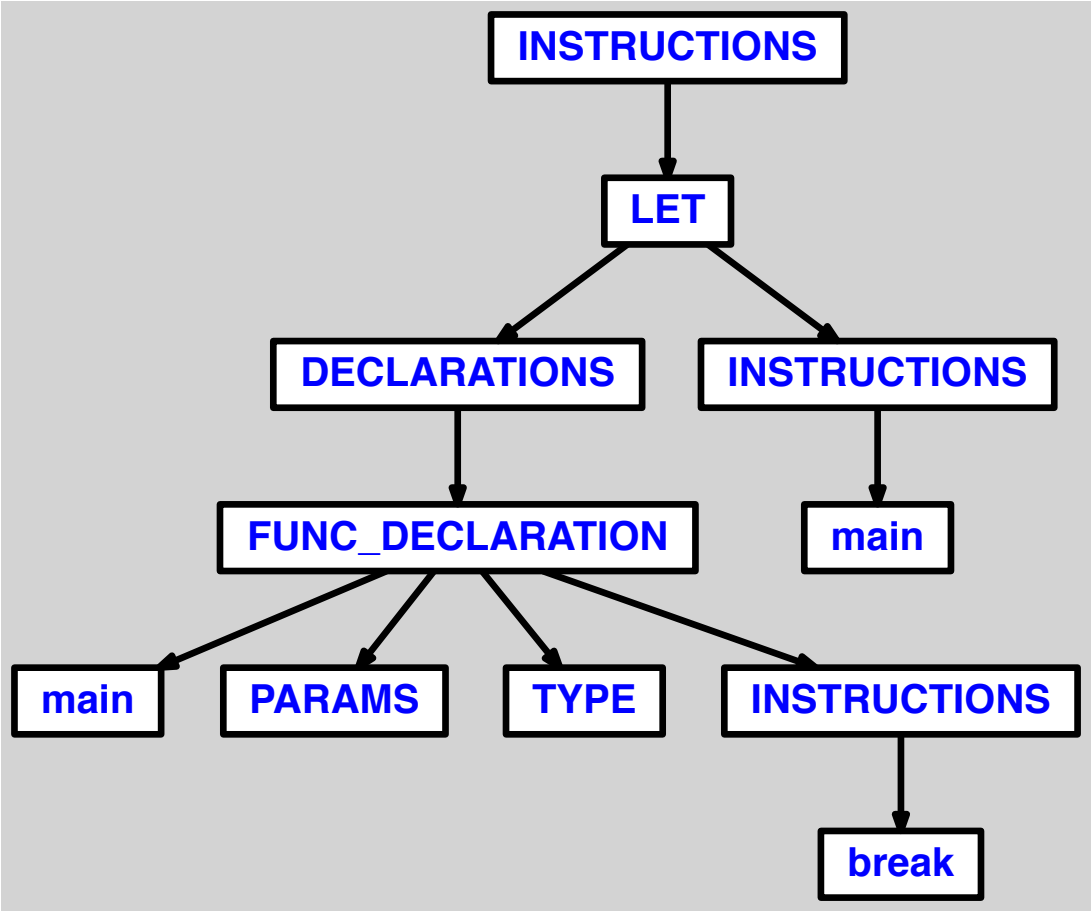
2.2.2 <break> dans un <for>, sans instruction avant

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
FOR
i
INT
INT
INSTRUCTIONS
0
3
break

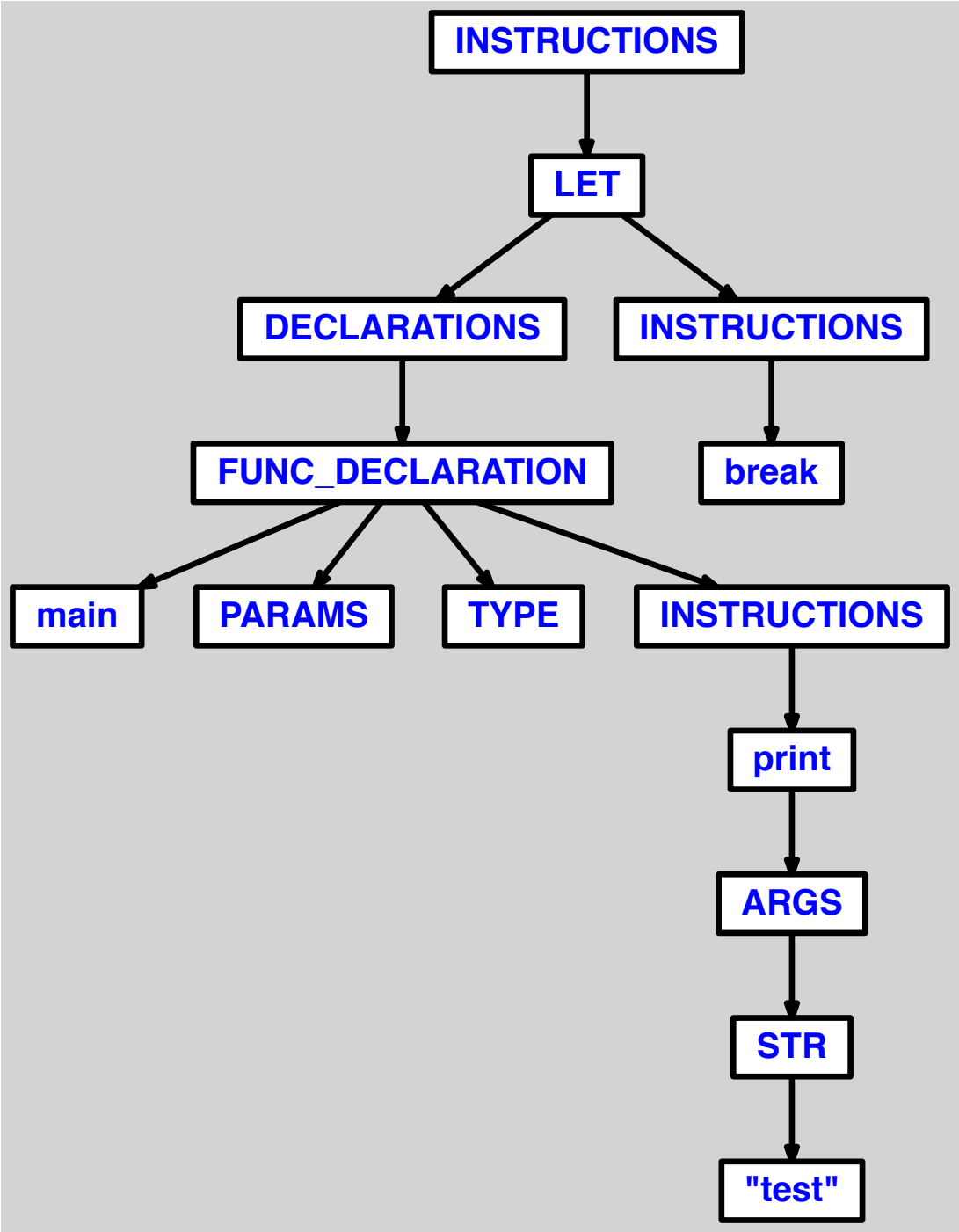
2.2.3 fonction executant seulement <break>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
break

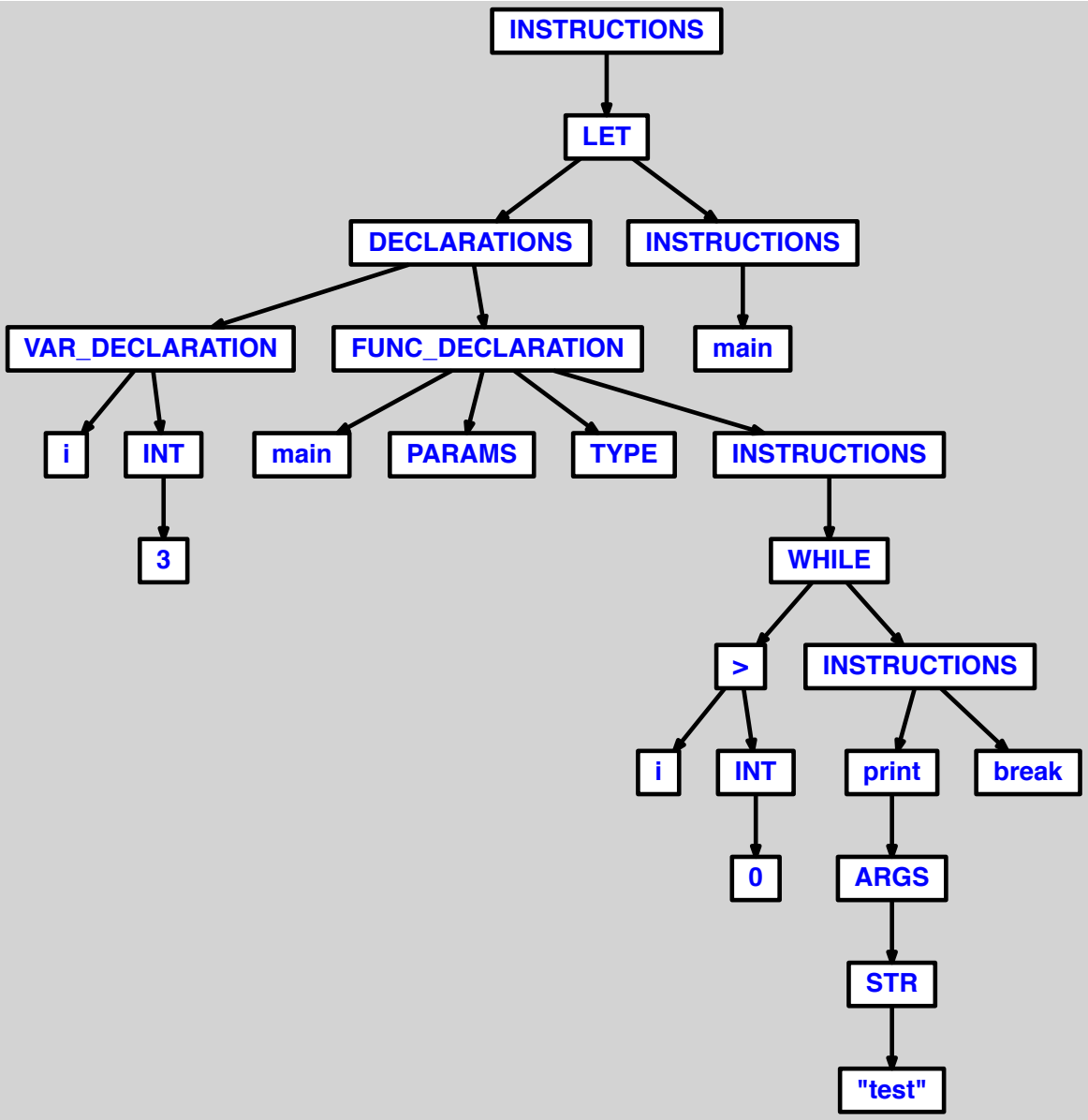
2.2.4 appel de <break>, avec corps de programme ne contenant pas <break>

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


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

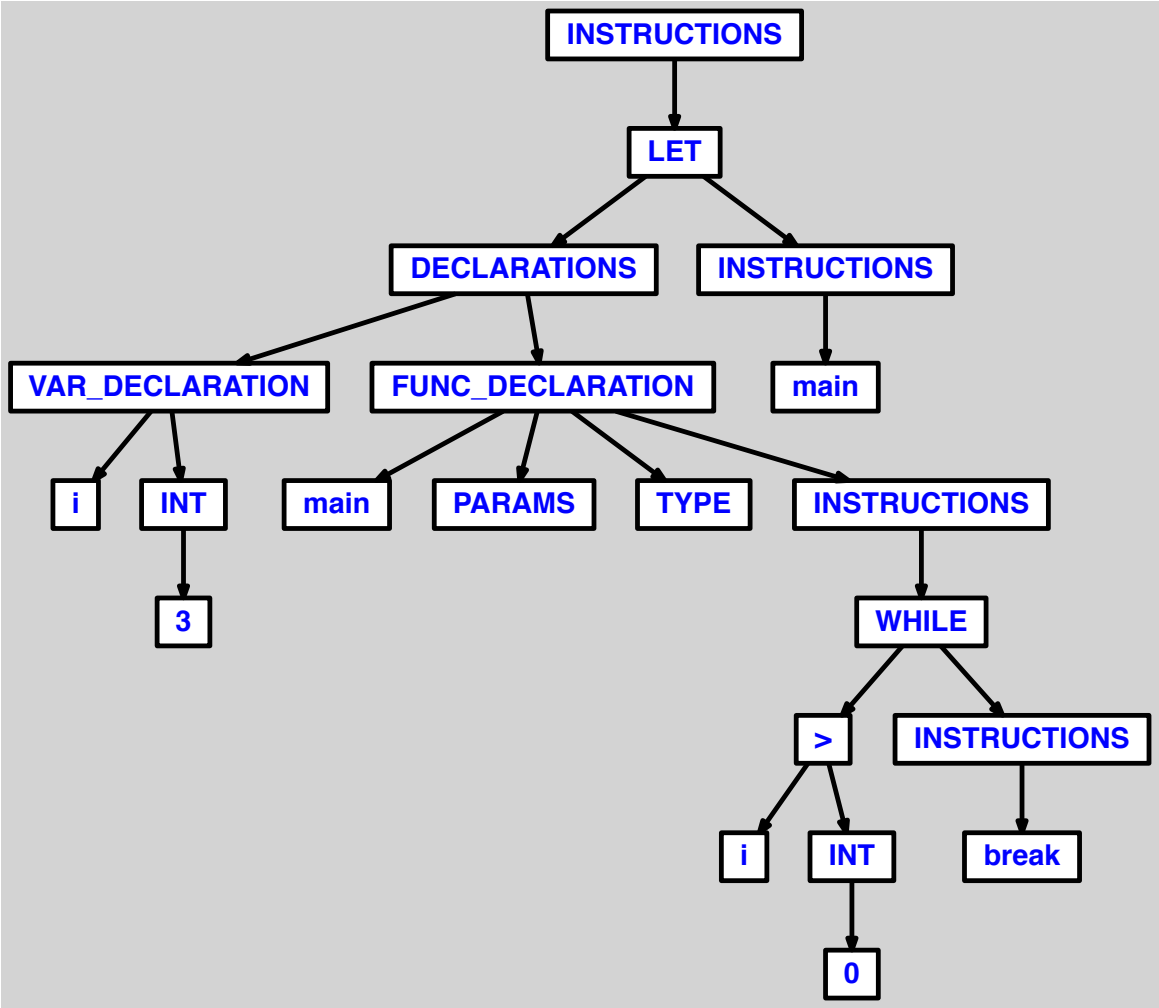
2.2.5 <break> dans un <while>, apres instruction

```
let
  var i := 3

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


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

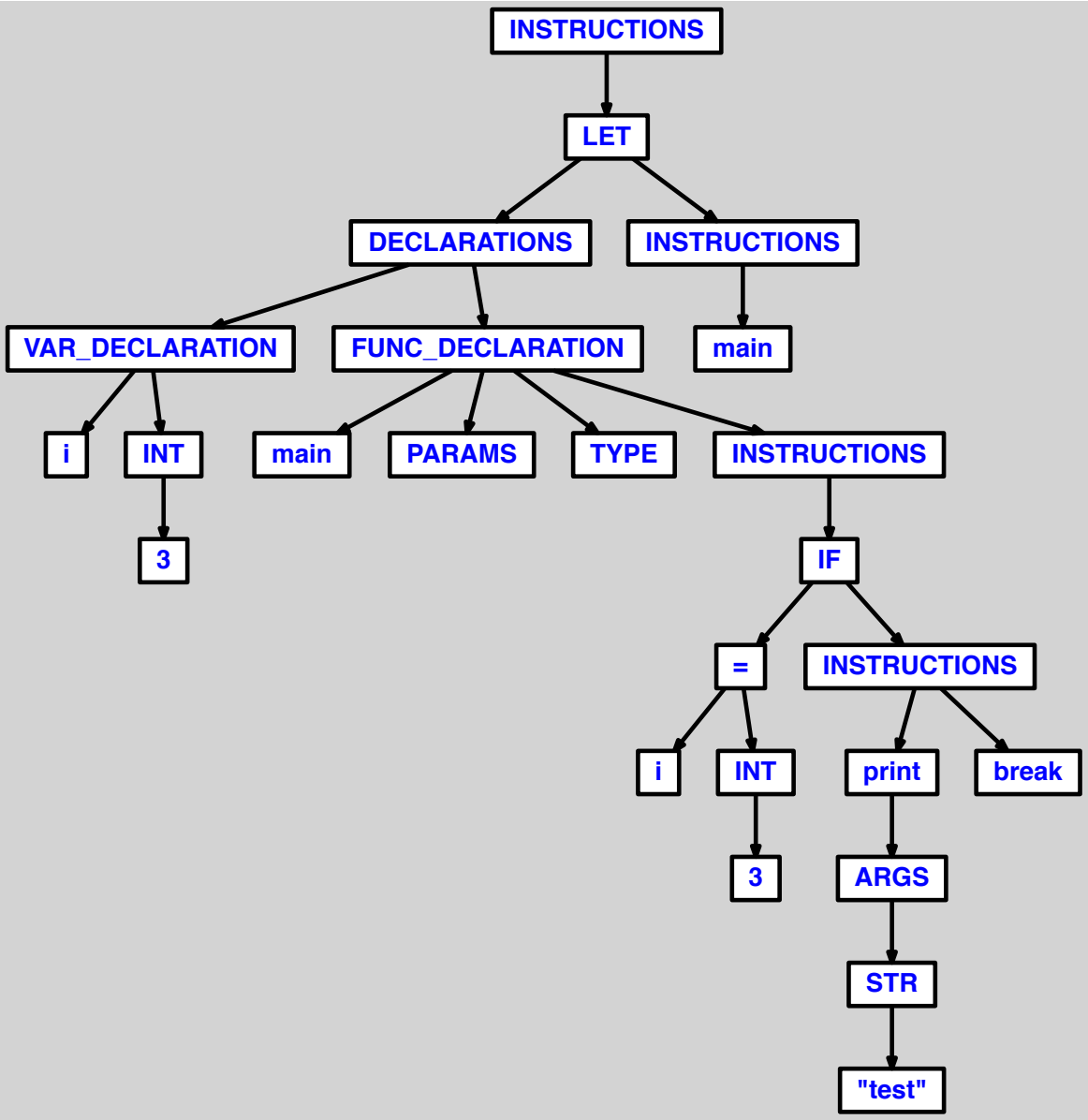
2.2.6 <break> dans un <while>, sans instruction avant

```
let
  var i := 3

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
main
i
INT
main
PARAMS
TYPE
INSTRUCTIONS
3
WHILE
>
INSTRUCTIONS
i
INT
break
0

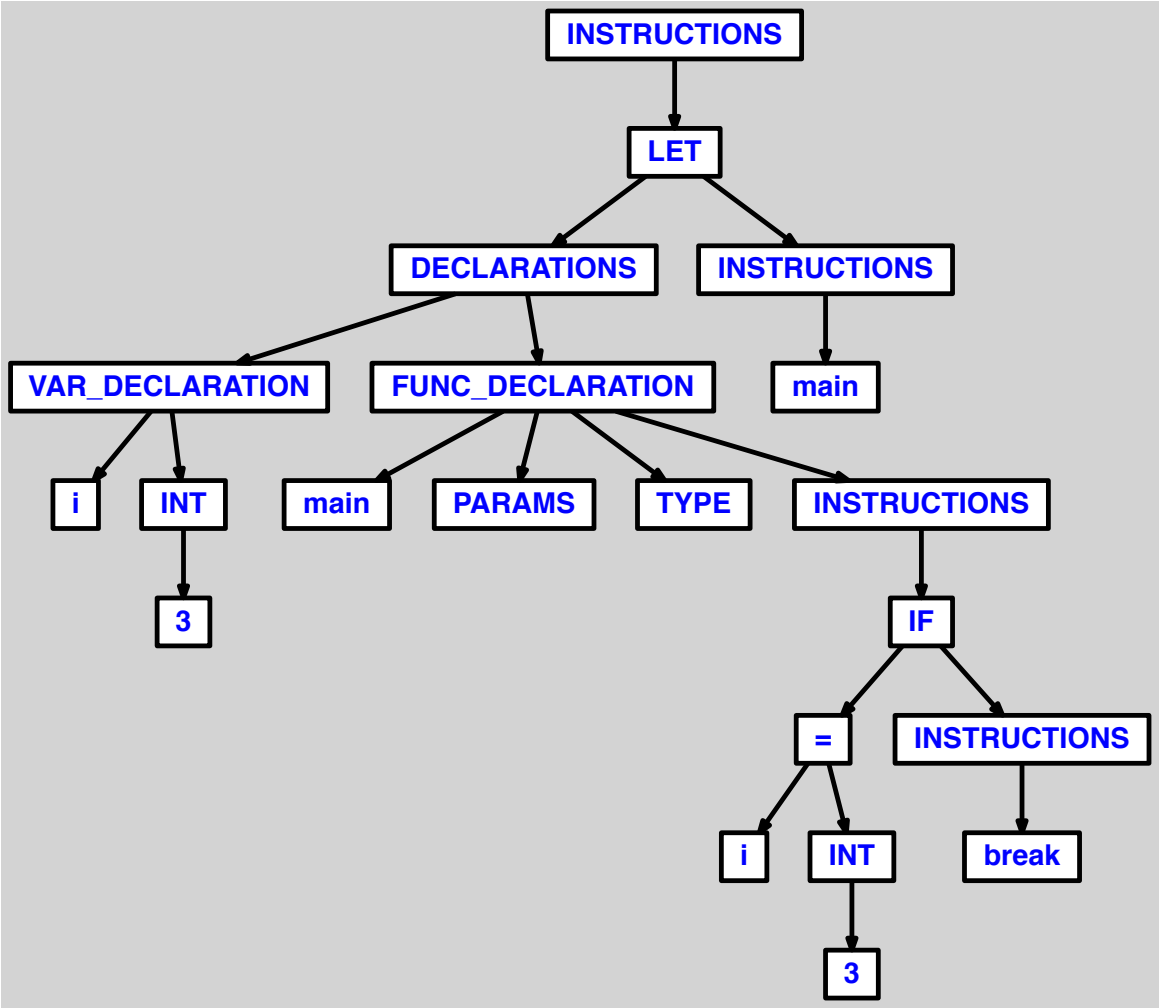
2.2.7 <break> dans un <if then>, apres instruction

```
let
  var i := 3

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


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

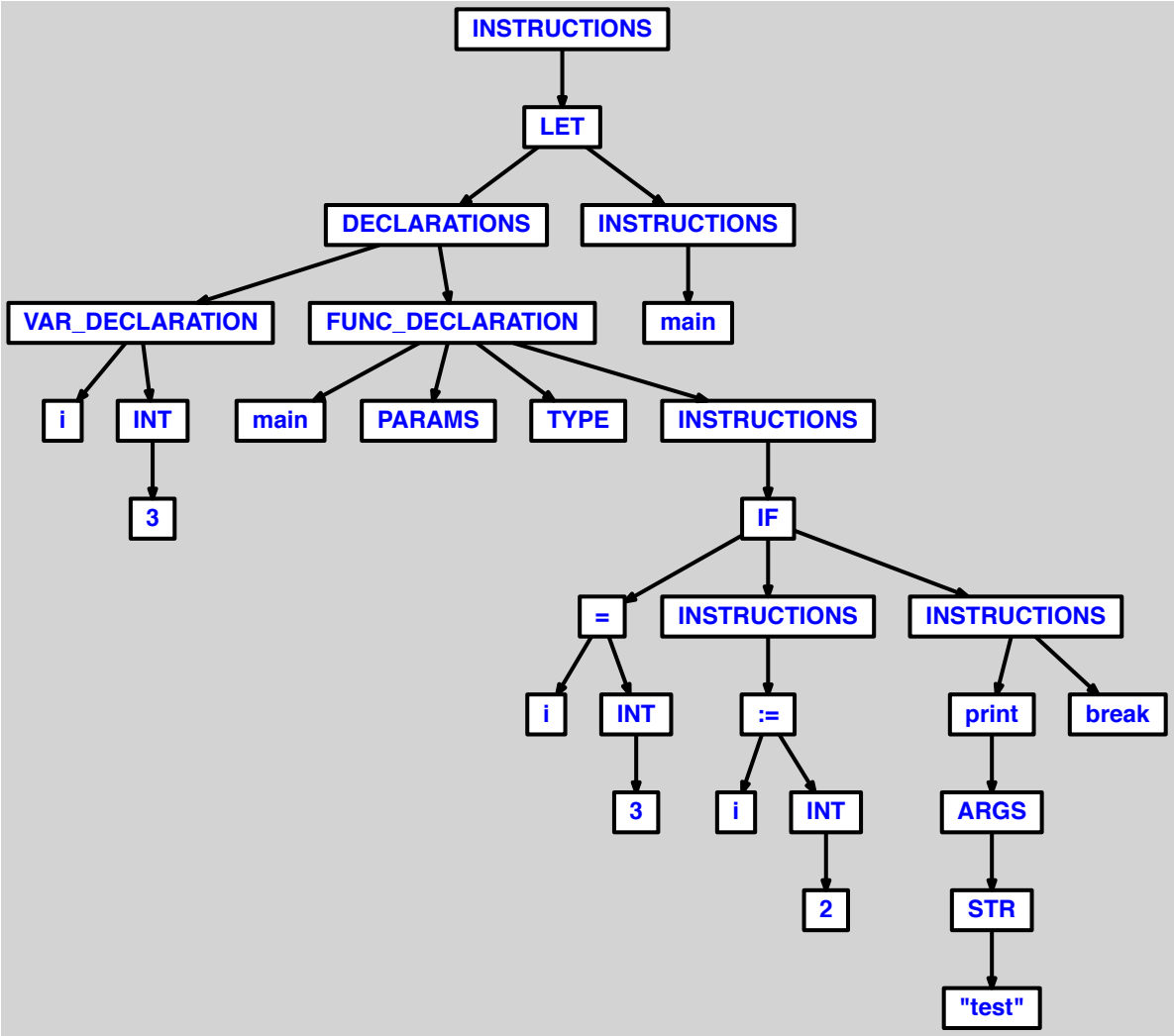
2.2.8 <break> dans un <if then>, sans instruction avant

```
let
  var i := 3

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
main
i
INT
main
PARAMS
TYPE
INSTRUCTIONS
3
IF
=
INSTRUCTIONS
i
INT
break
3

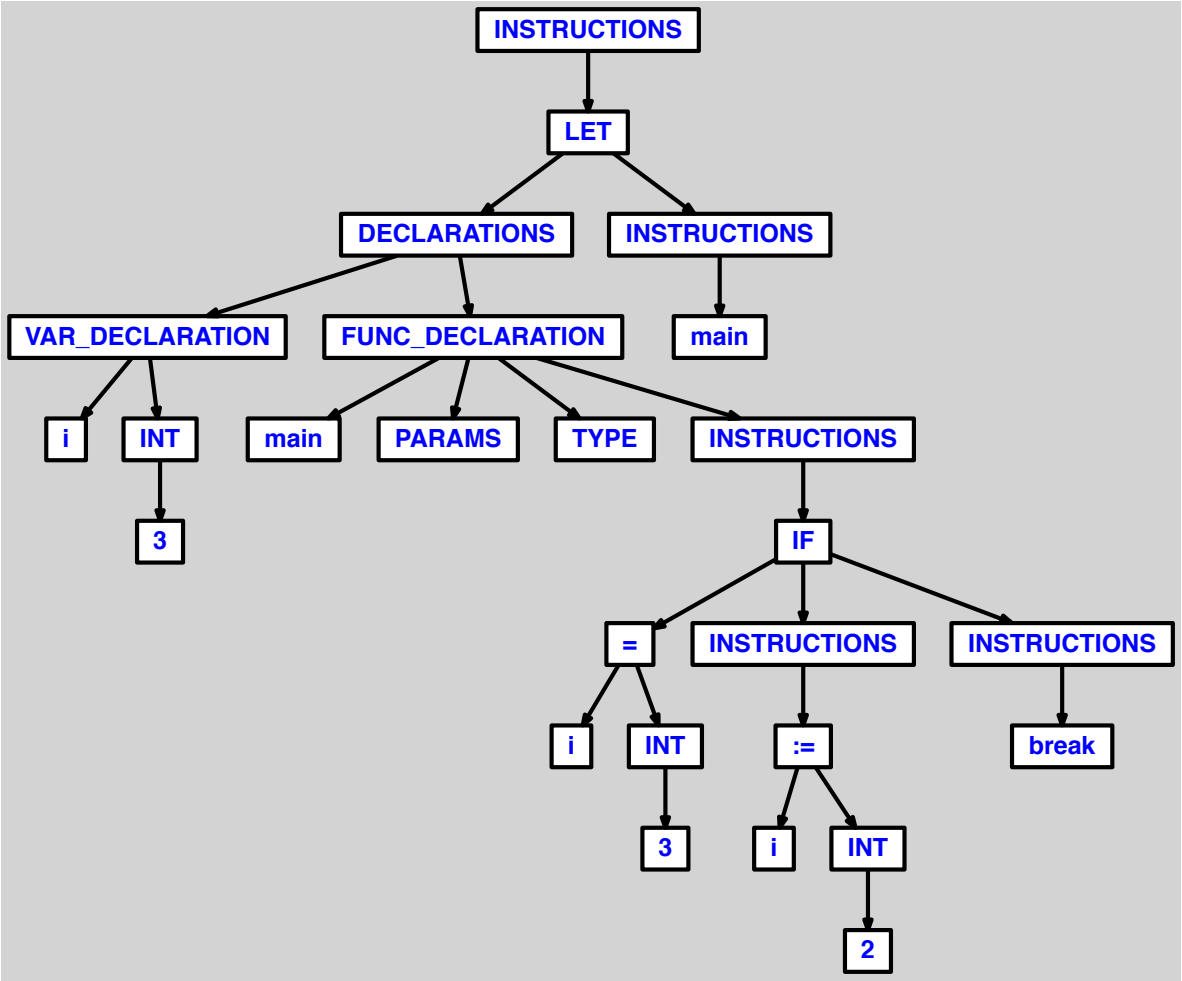
2.2.9 <break> dans un <if then else>, apres instruction

```
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
main
i
INT
main
PARAMS
TYPE
INSTRUCTIONS
3
IF
=
INSTRUCTIONS
INSTRUCTIONS
i
INT
:=
print
break
3
i
INT
ARGS
2
STR
"test"

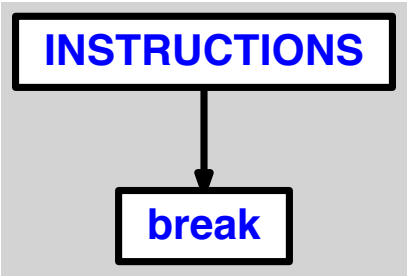
2.2.10 <break> dans un <if then else>, sans instruction avant

```
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
main
i
INT
main
PARAMS
TYPE
INSTRUCTIONS
3
IF
=
INSTRUCTIONS
INSTRUCTIONS
i
INT
:=
break
3
i
INT
2

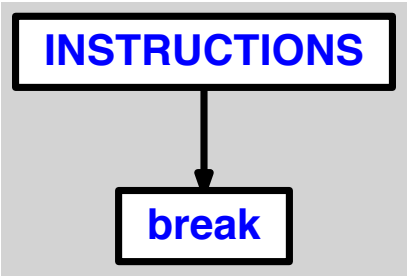
2.2.11 <break> seul

```
break
```


INSTRUCTIONS
break

2.2.12 <break> seul, apres saut de ligne

```

break
```

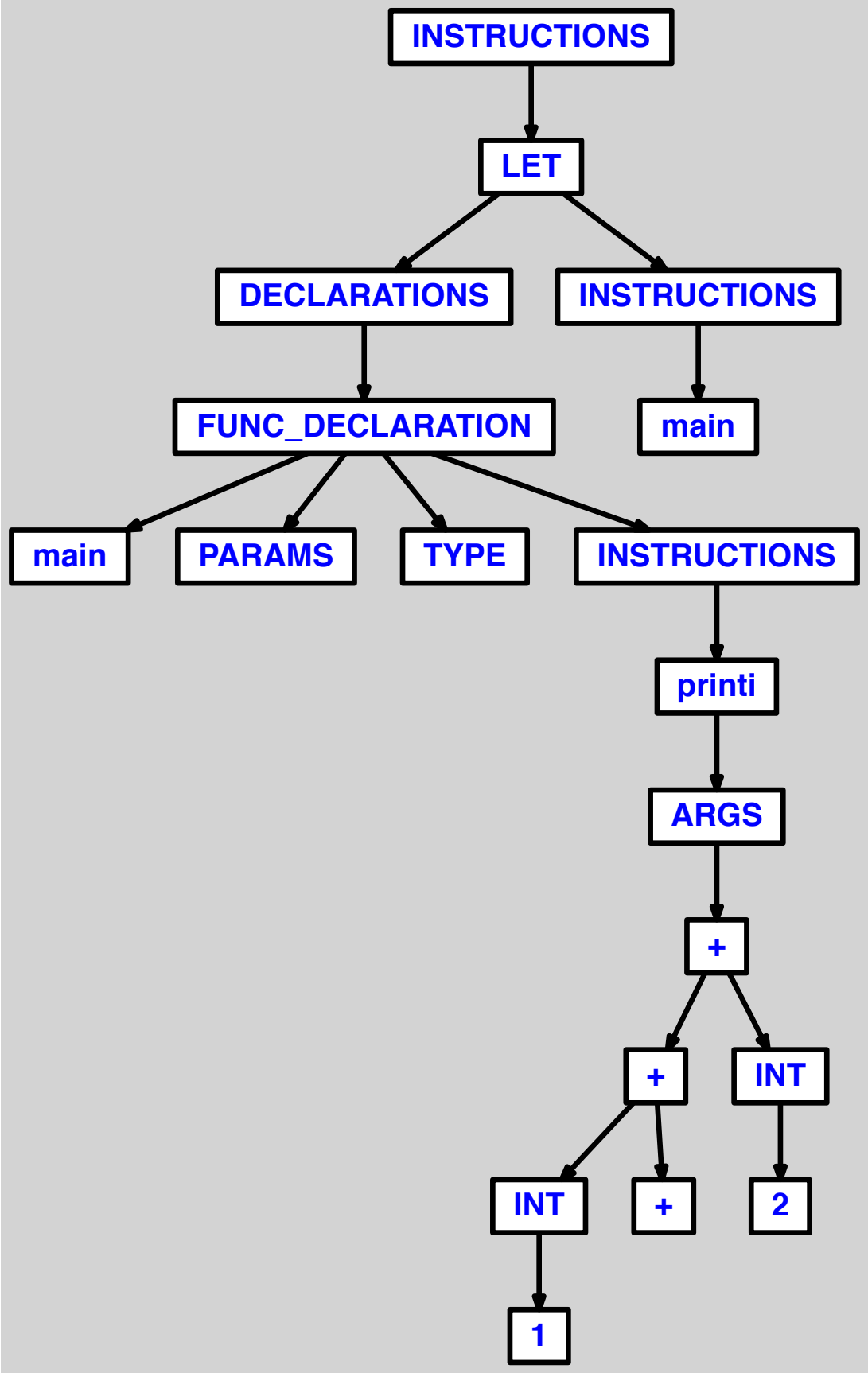

INSTRUCTIONS
break

3 calcul

3.1 KO

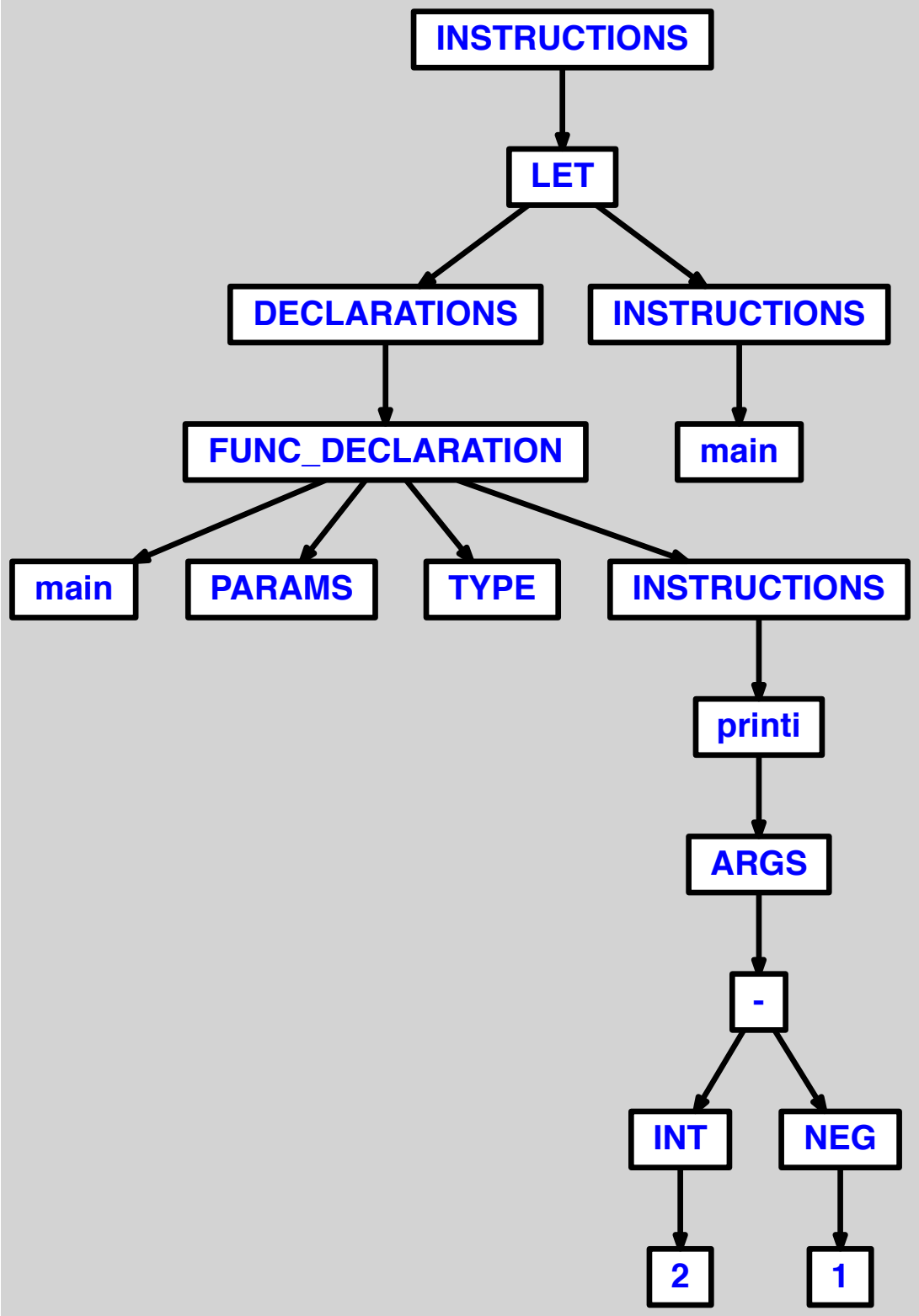
3.1.1 addition avec operateur mal ecrit

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


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

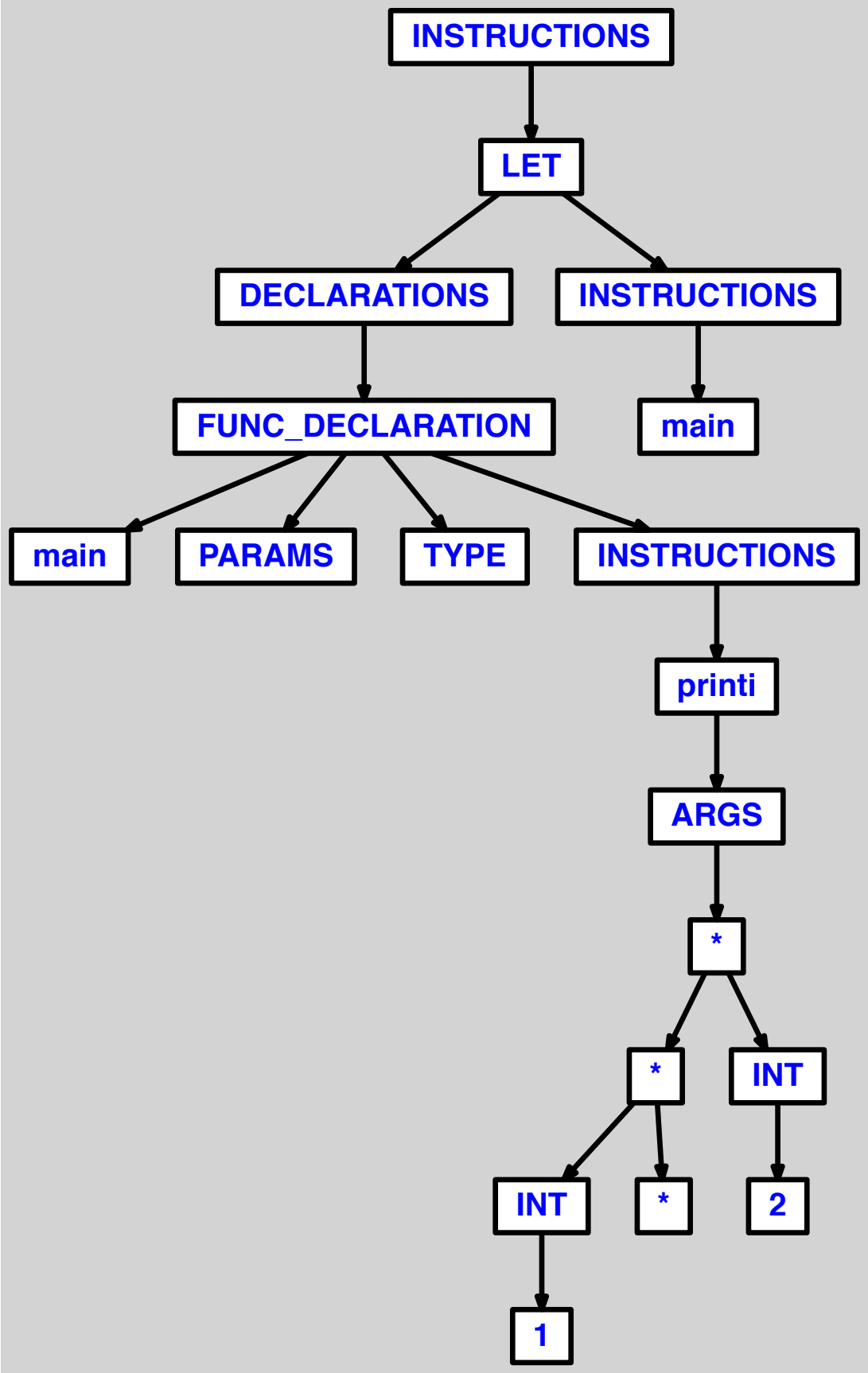
3.1.2 soustraction avec operateur mal ecrit

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


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

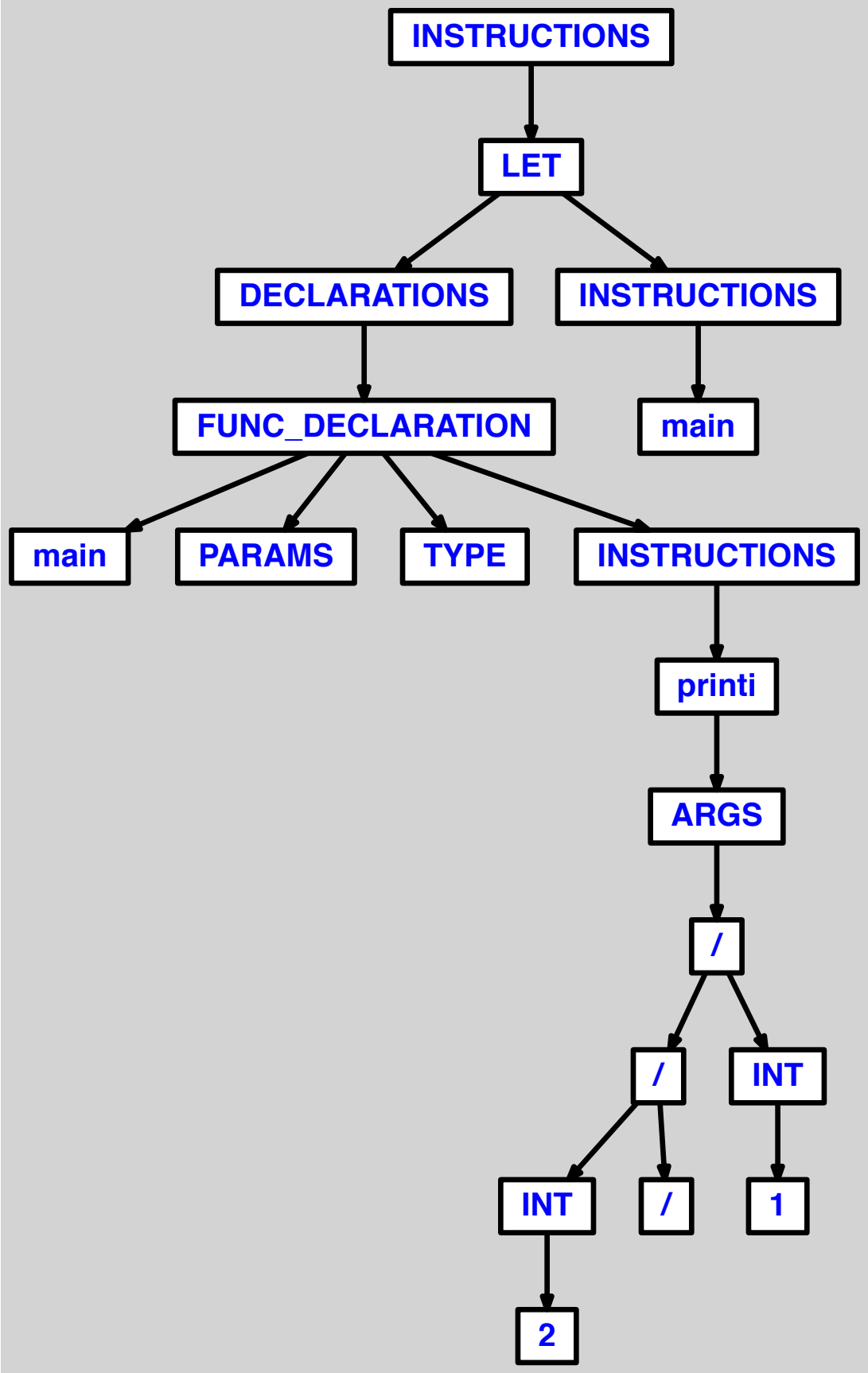
3.1.3 multiplication avec operateur mal ecrit

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


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

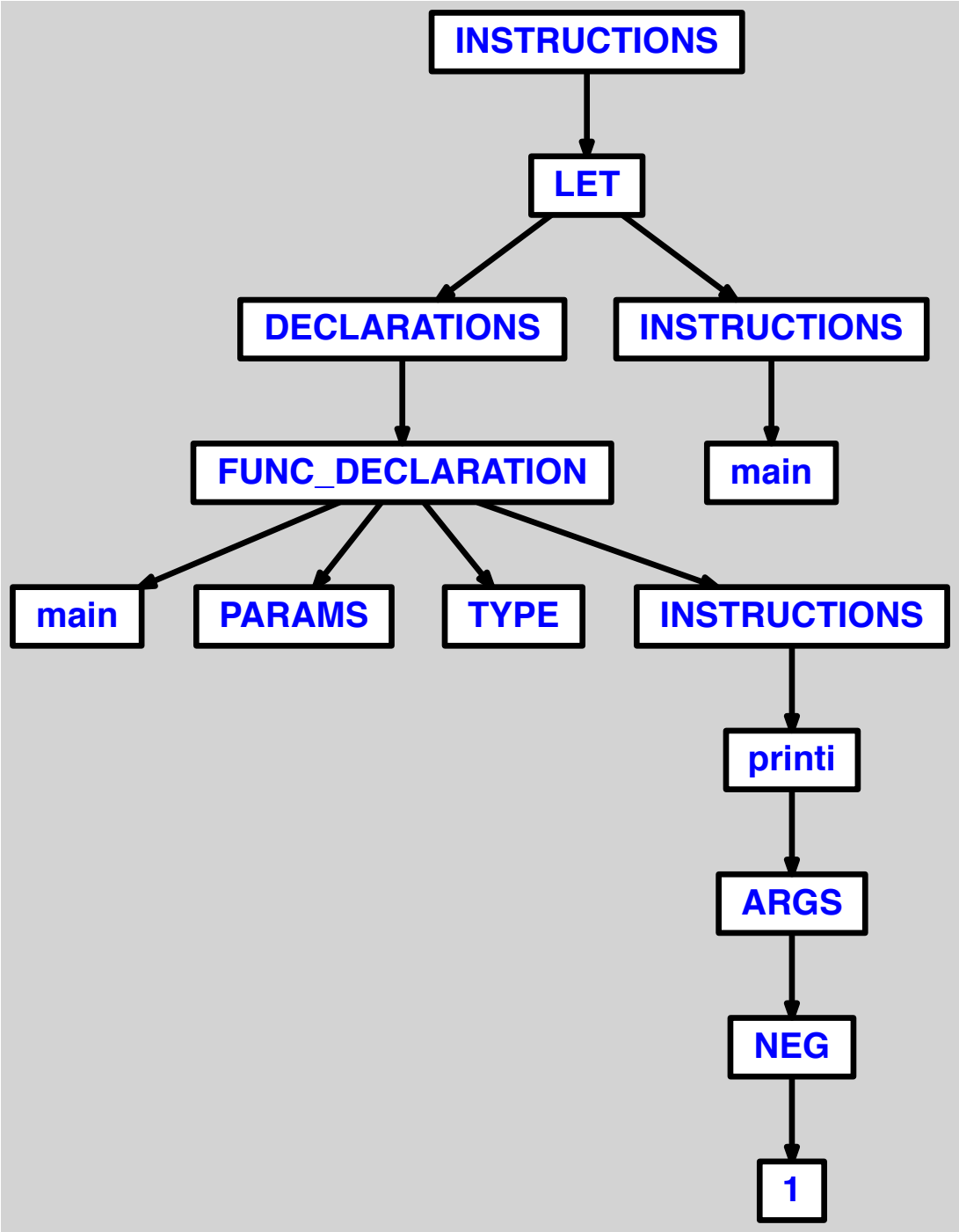
3.1.4 division avec operateur mal ecrit

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


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

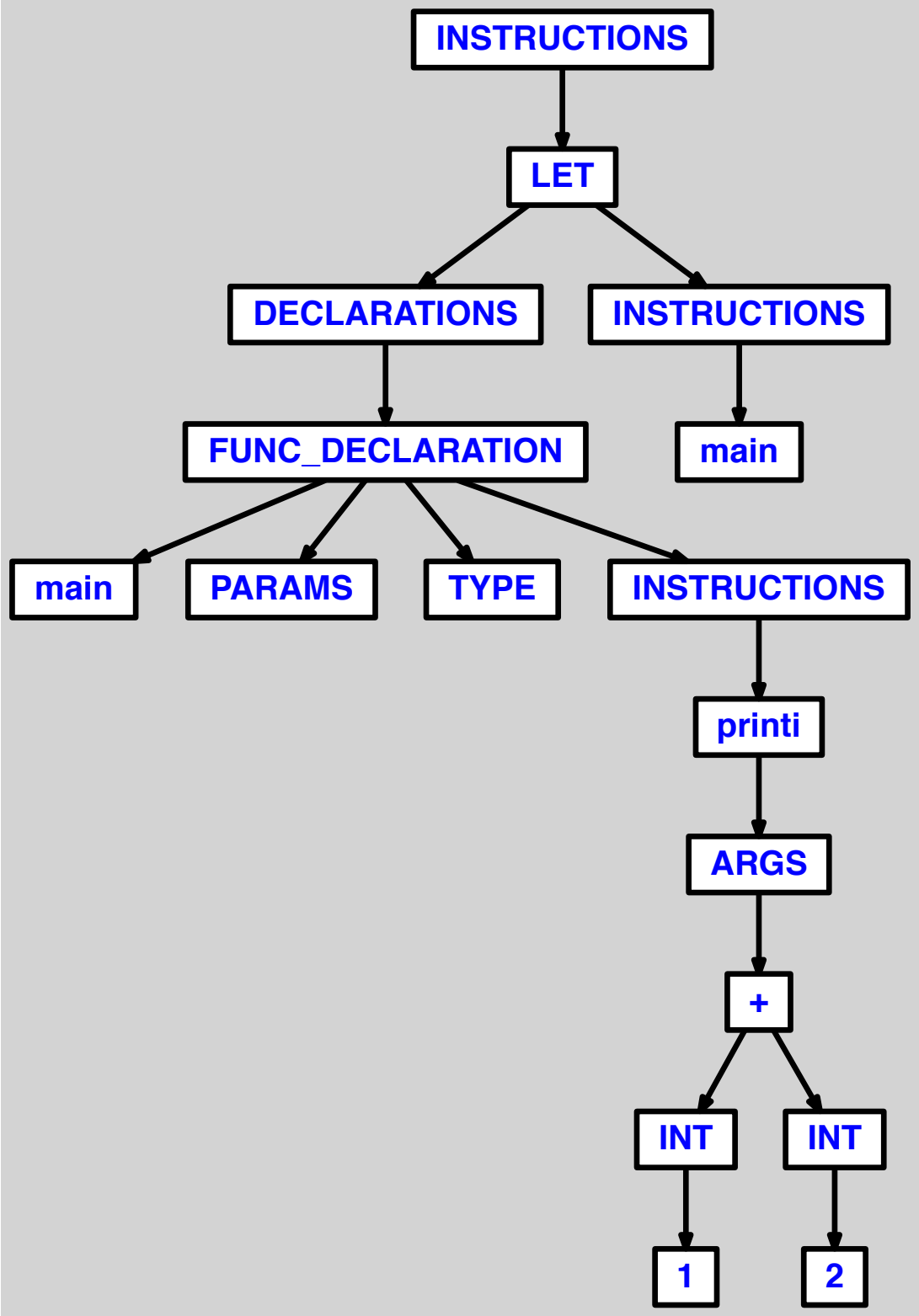
3.1.5 moins unaire avec operateur mal ecrit

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
printi
ARGS
NEG
1

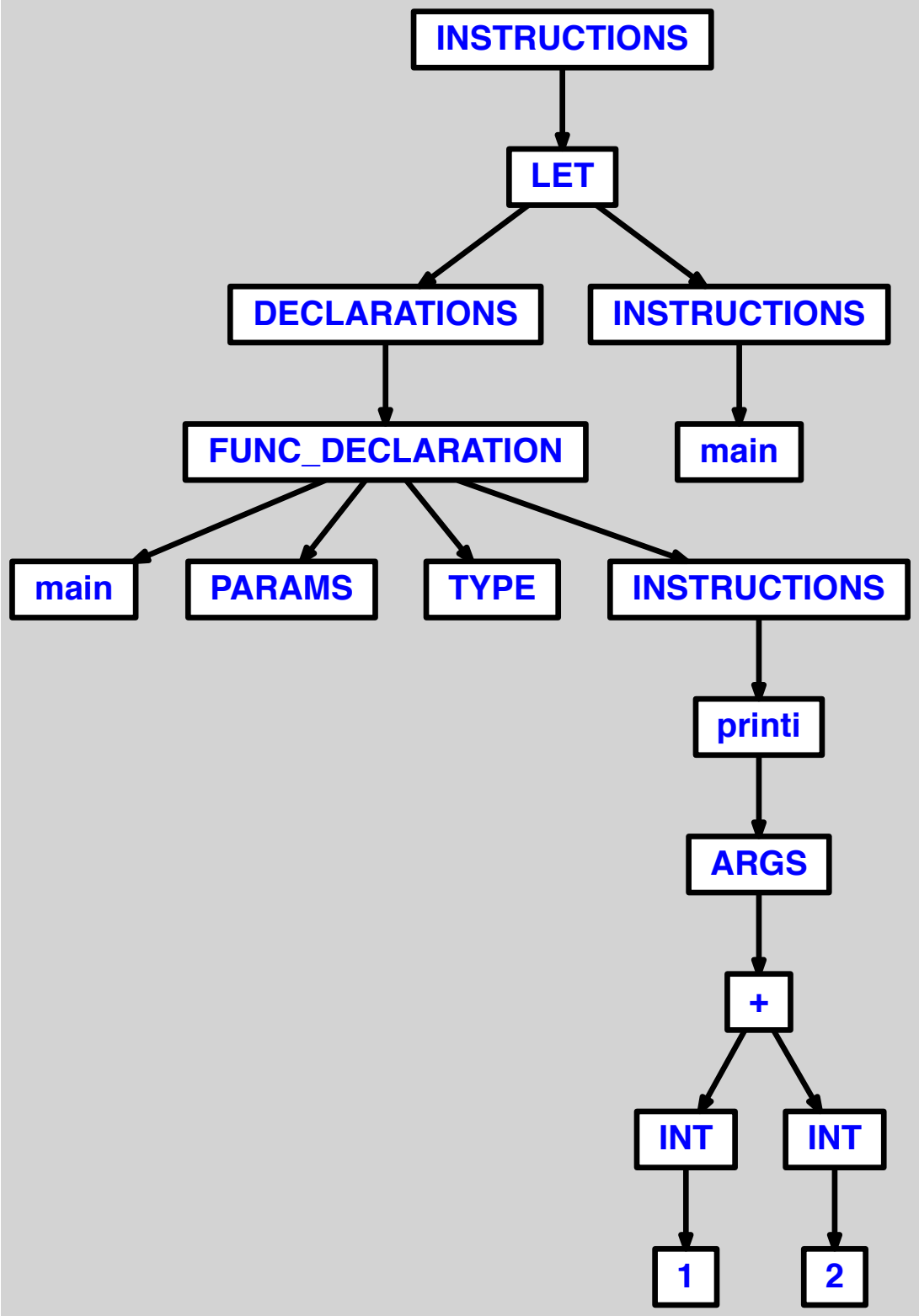
3.1.6 calcul a 2 termes avec oubli de parenthese gauche

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


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

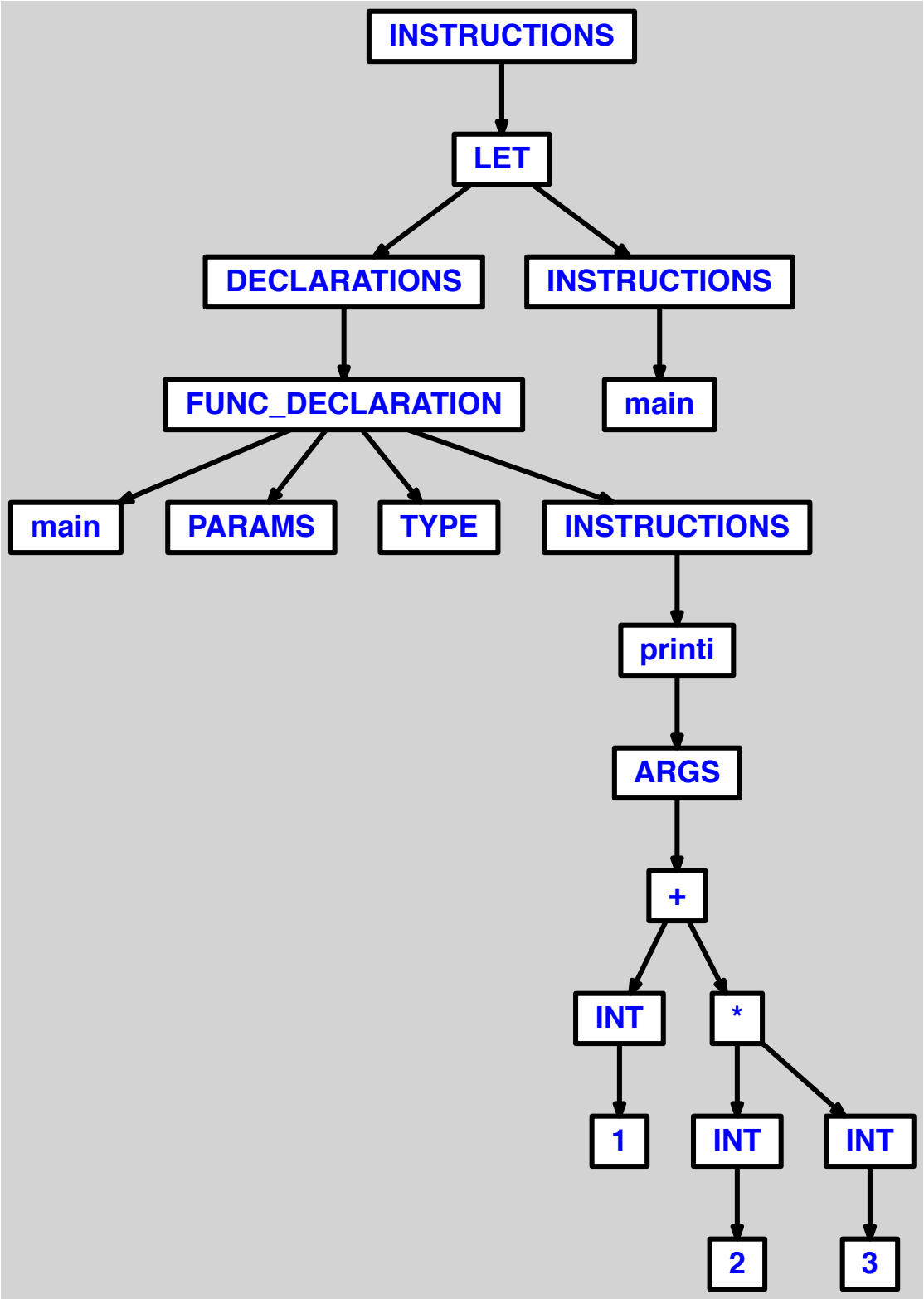
3.1.7 calcul a 2 termes avec oubli de parenthese droite

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


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

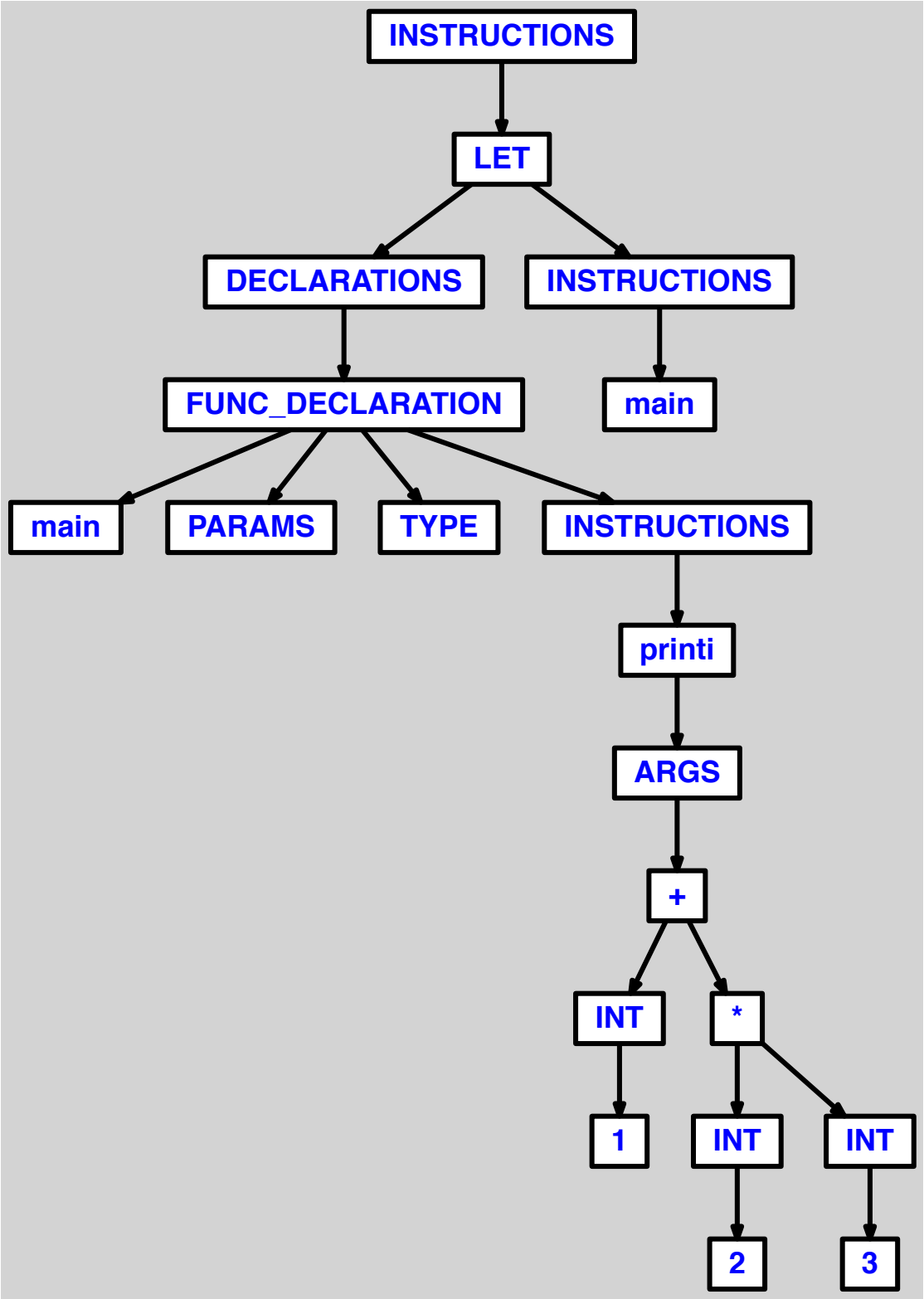
3.1.8 calcul a 3 termes avec oubli de parenthese gauche

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


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

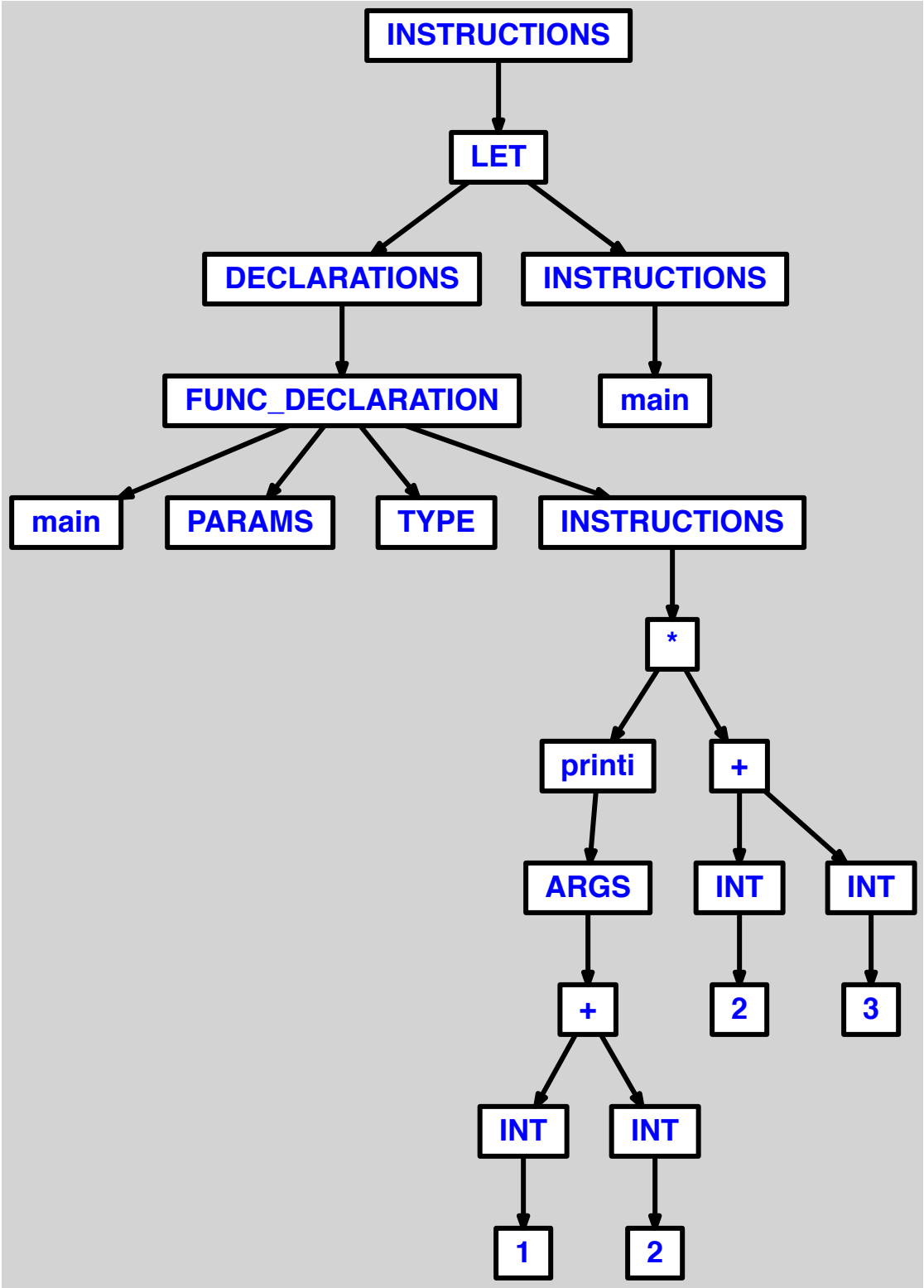
3.1.9 calcul a 3 termes avec oubli de parenthese droite

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


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

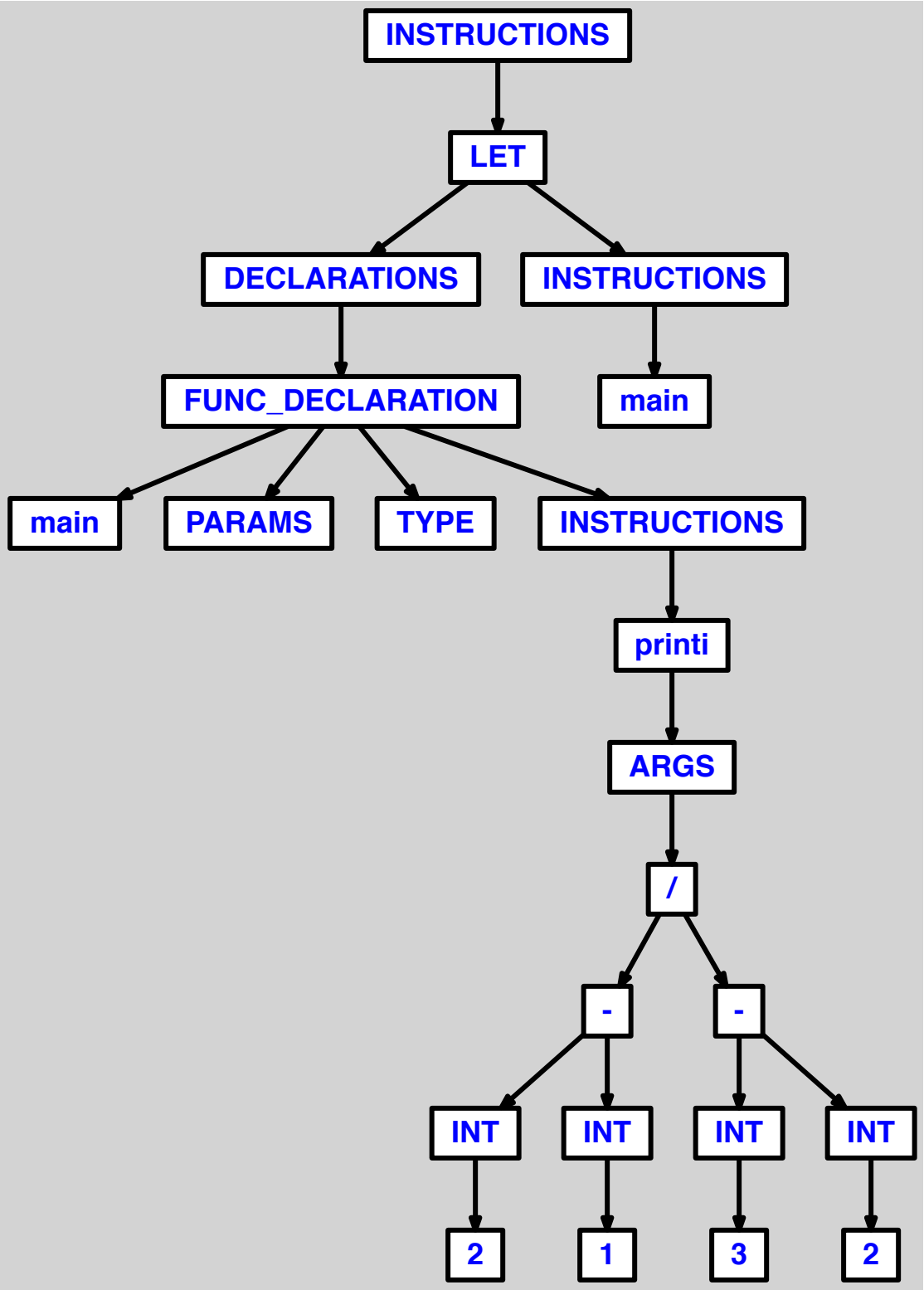
3.1.10 calcul a 4 termes avec oubli de parenthese gauche

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


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

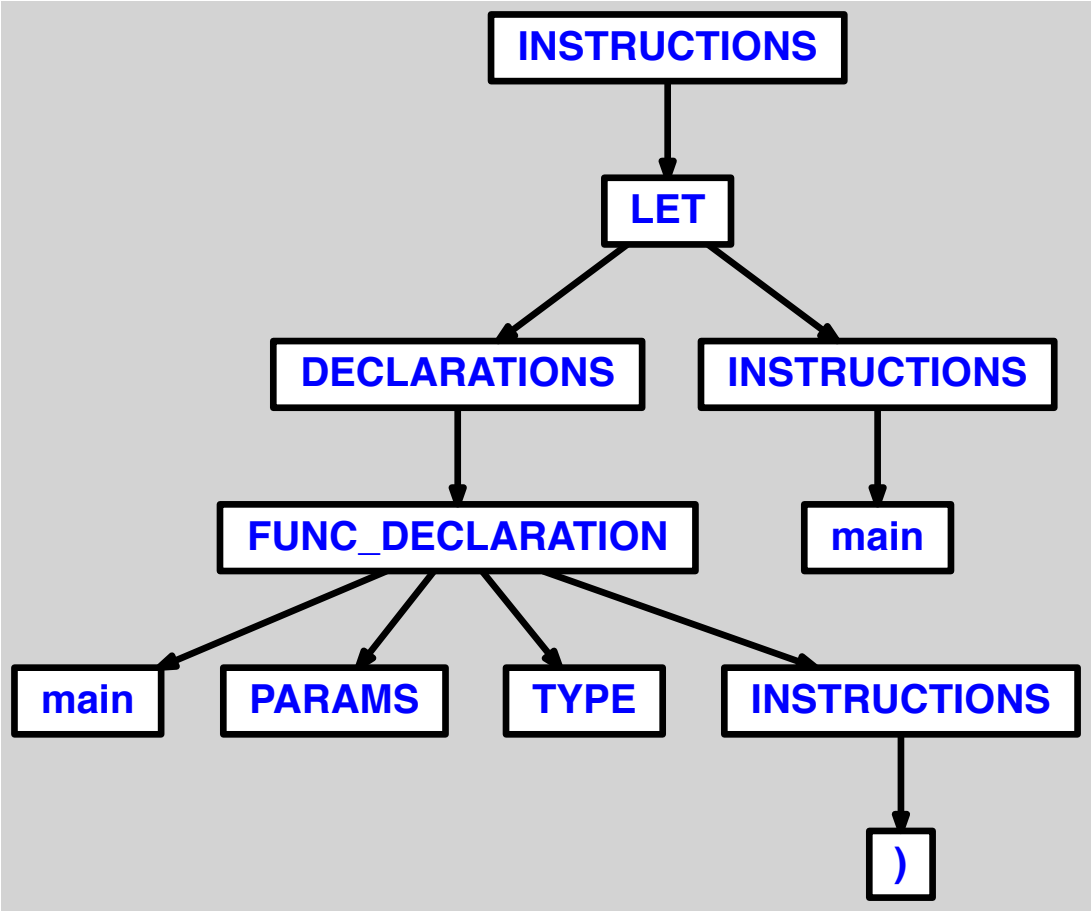
3.1.11 calcul a 4 termes avec oubli de parenthese droite

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


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

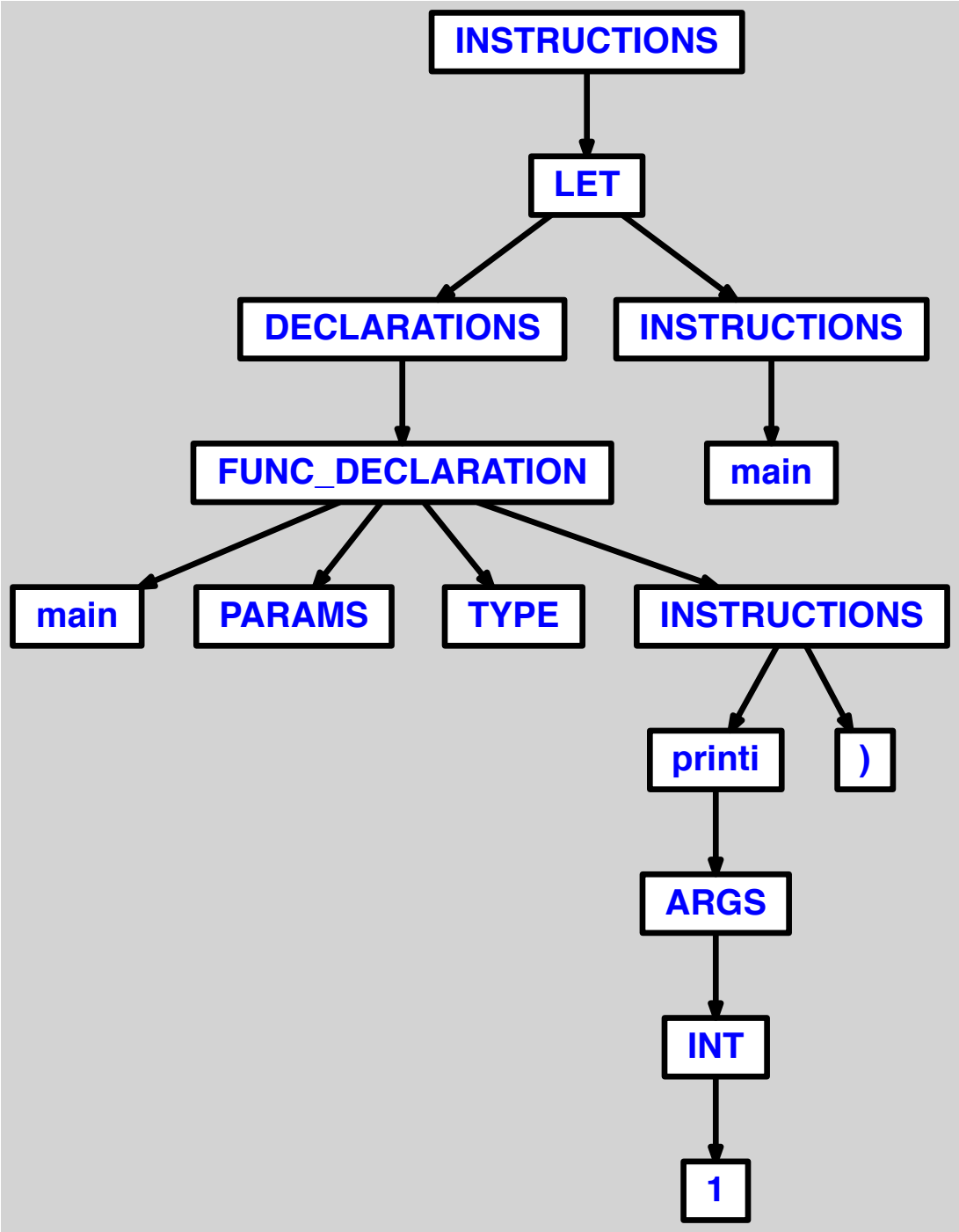
3.1.12 parenthesage sans contenu

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
)

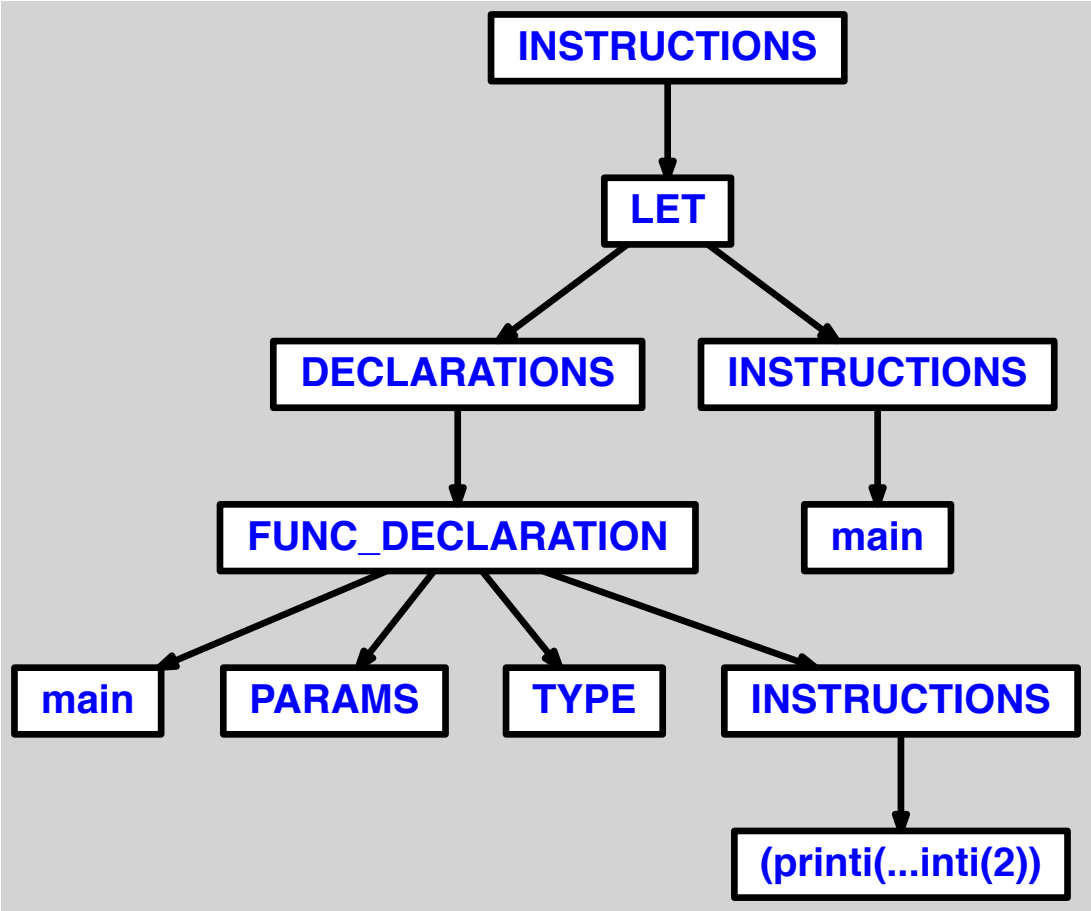
3.1.13 expression parenthesee avec <;> en trop

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


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

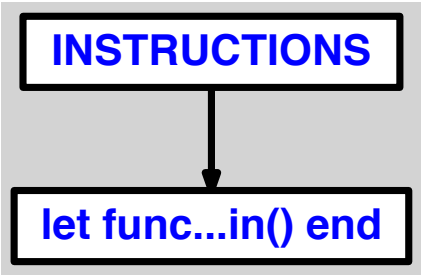
3.1.14 expression parenthesee avec oubli de <;>

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


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

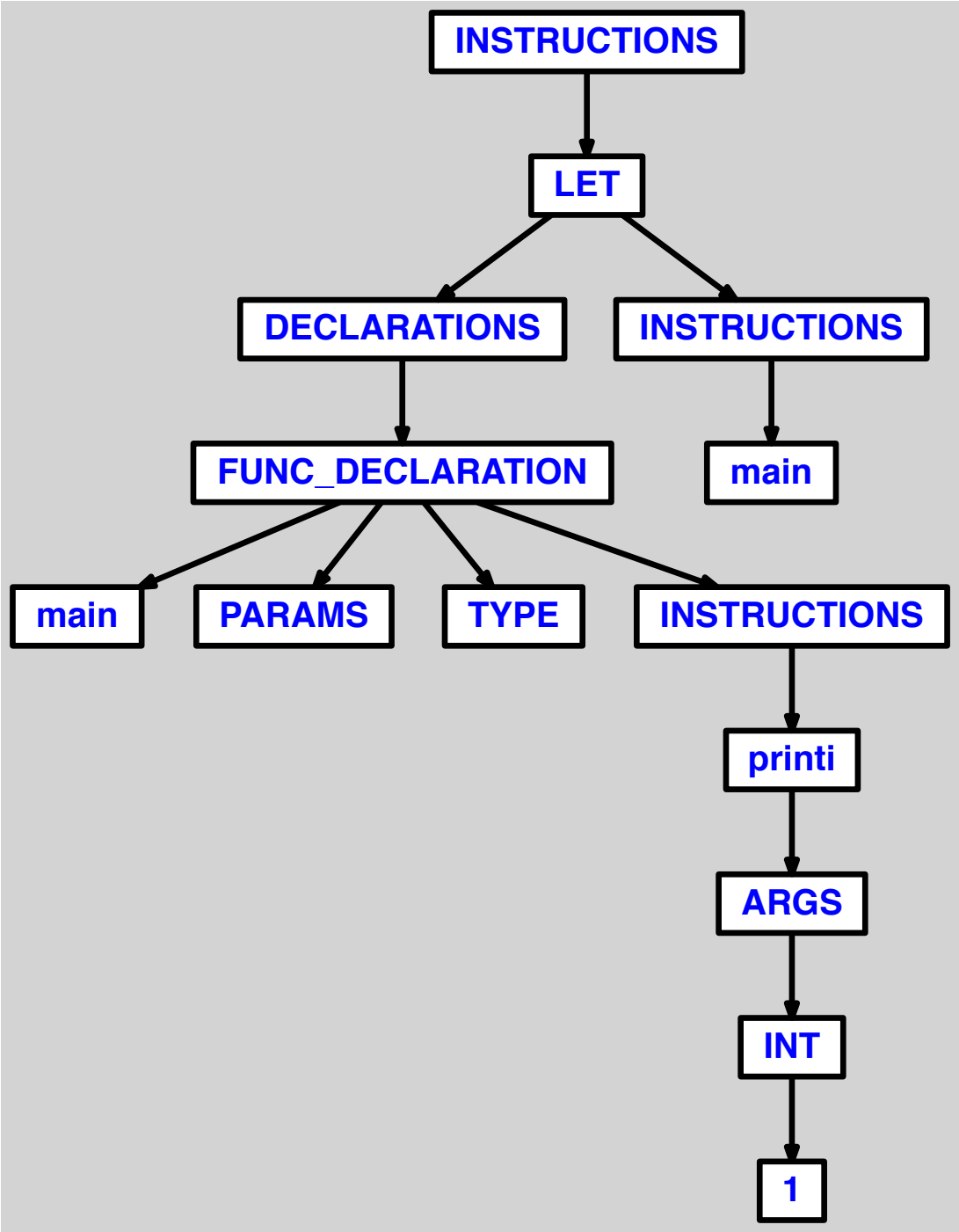
3.1.15 expression non parenthesee avec oubli de <;>

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


INSTRUCTIONS
let func...in() end

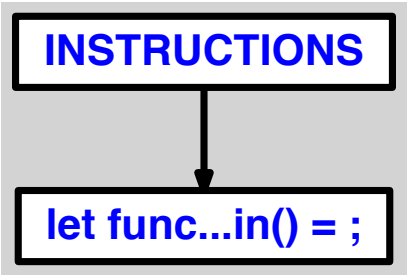
3.1.16 expression non parenthesee avec <;> en trop

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


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

3.1.17 <;> sans instruction avant

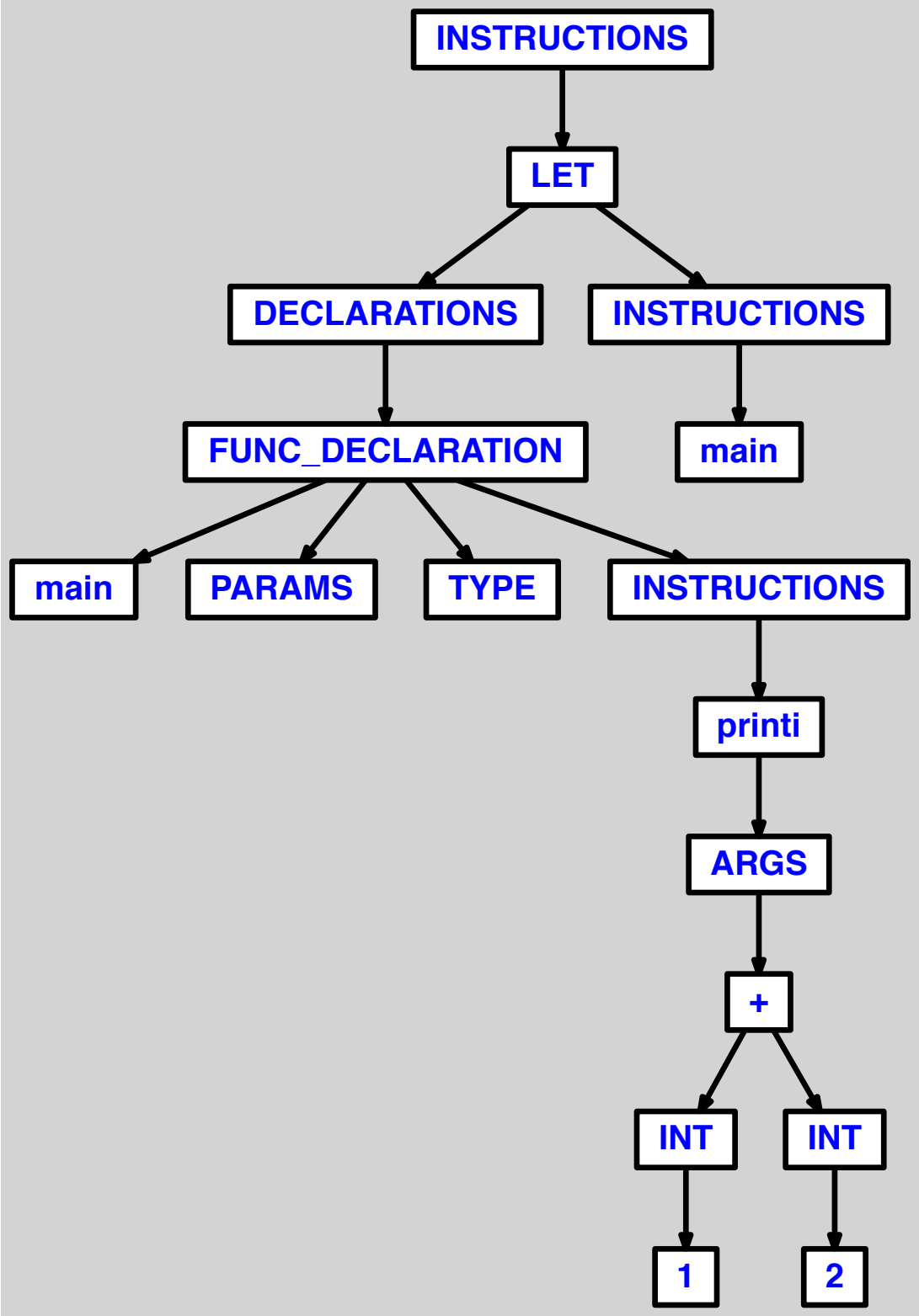
```
let function main() = ;
```


INSTRUCTIONS
let func...in() = ;

3.2 OK

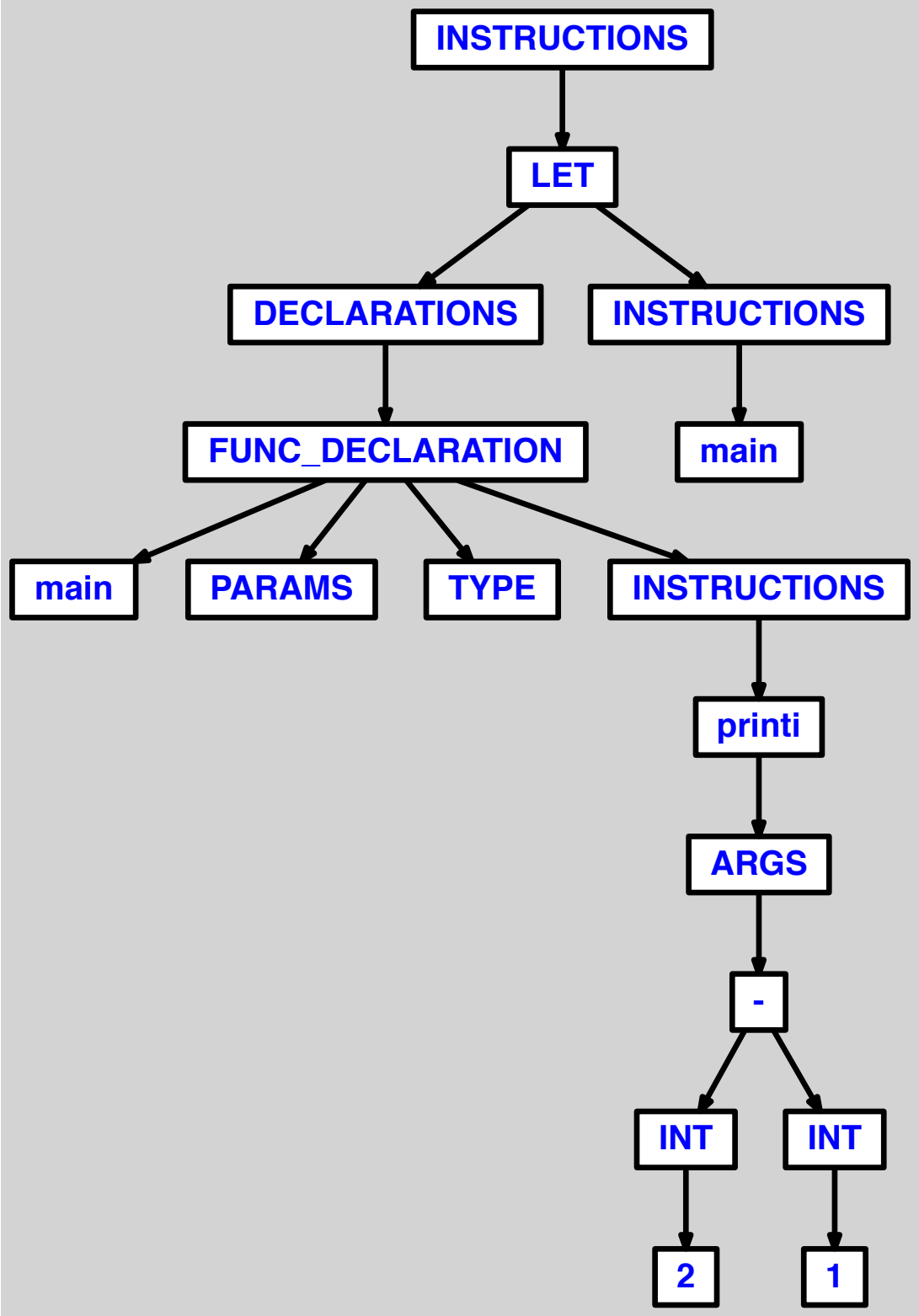
3.2.1 addition simple, a 2 termes

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


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

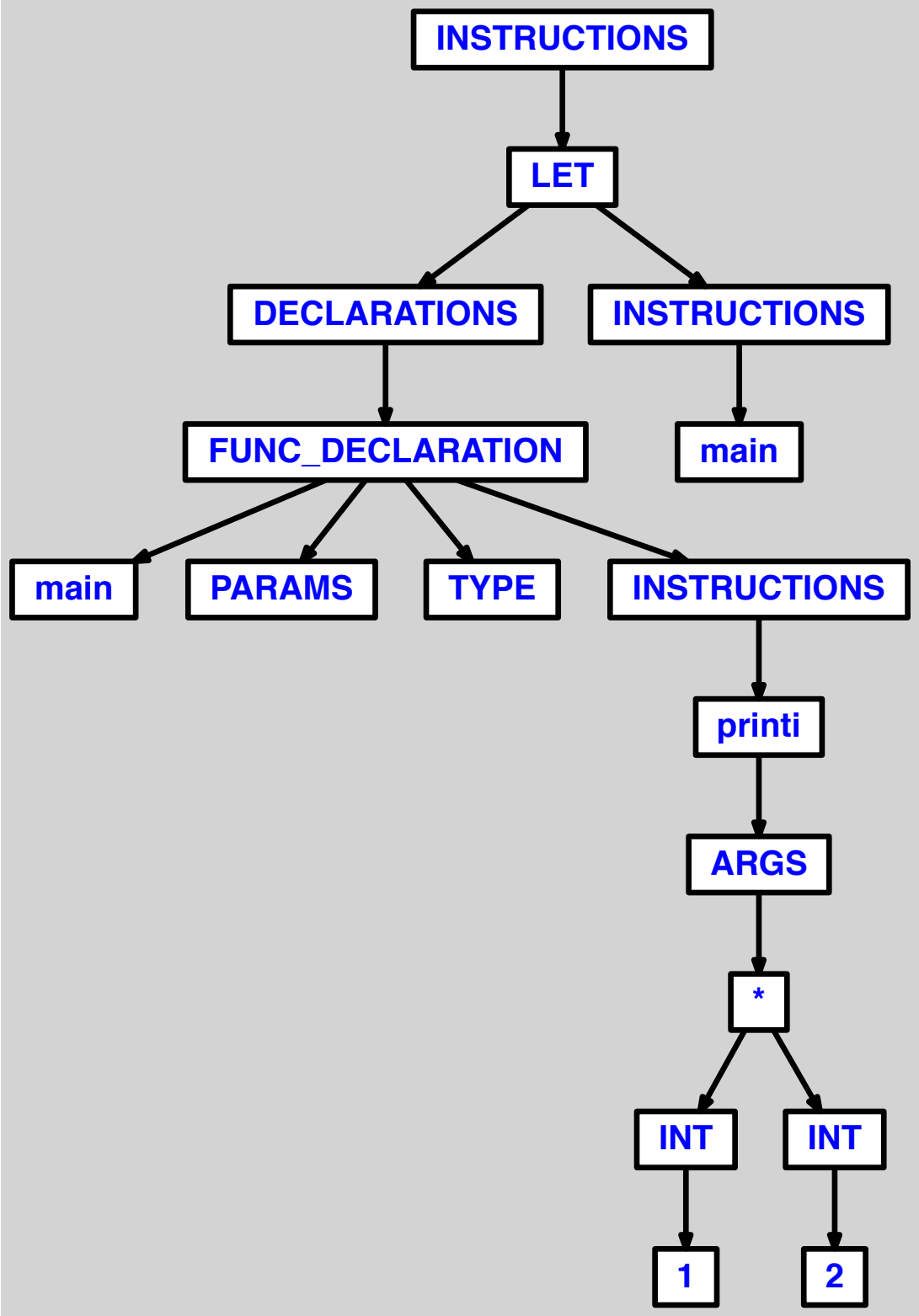
3.2.2 soustraction simple, a 2 termes

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


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

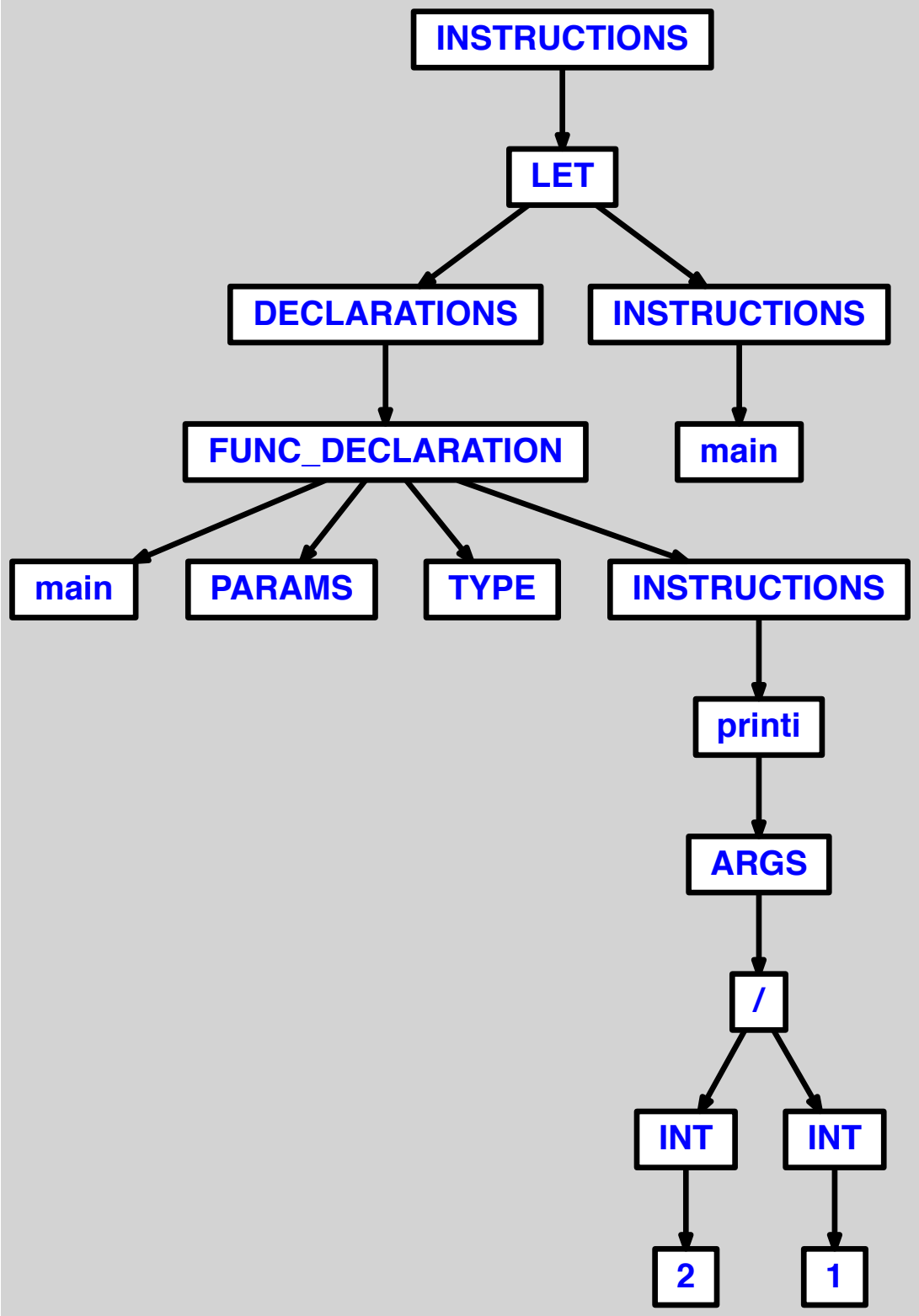
3.2.3 multiplication simple, a 2 termes

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


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

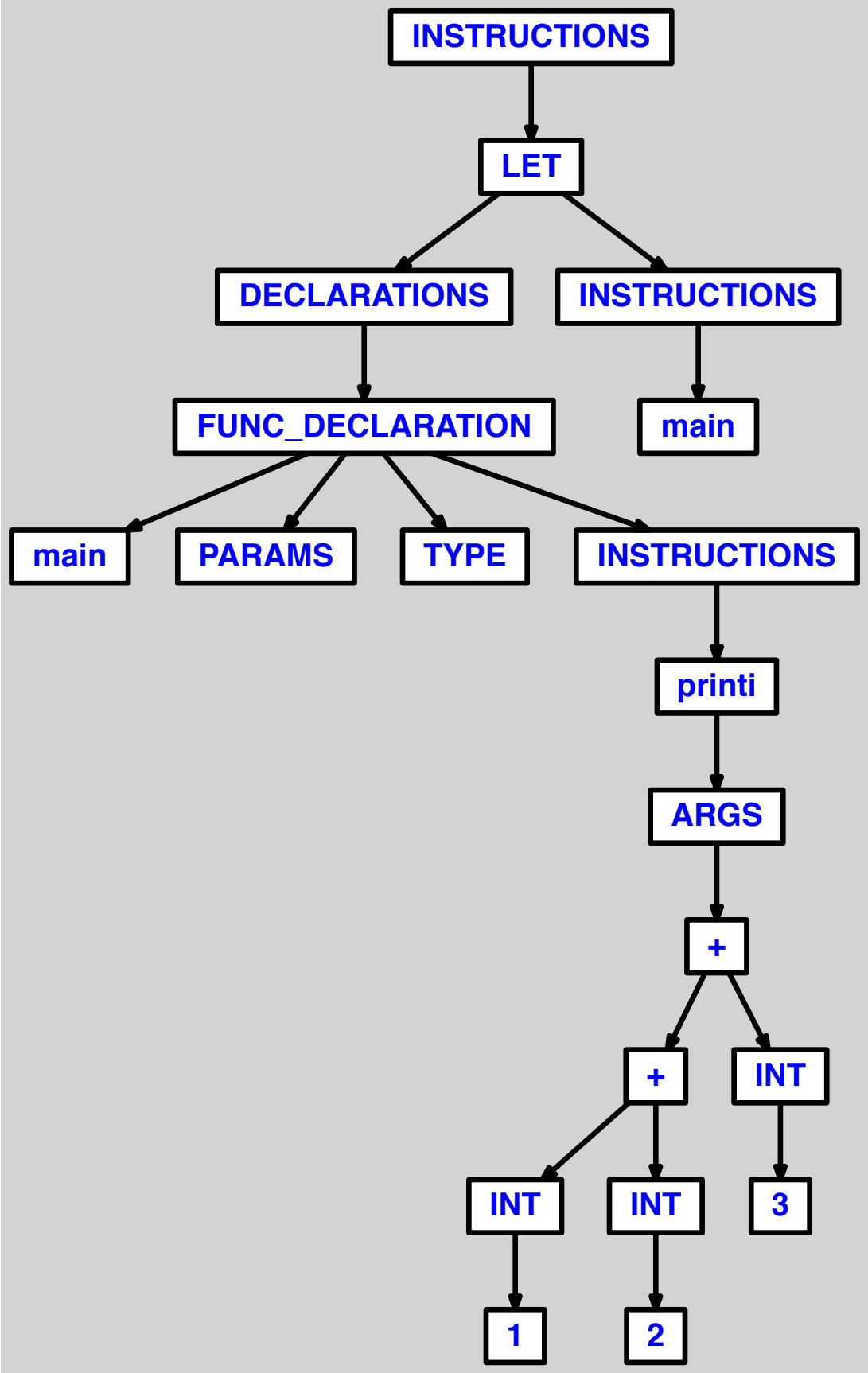
3.2.4 division simple, a 2 termes

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


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

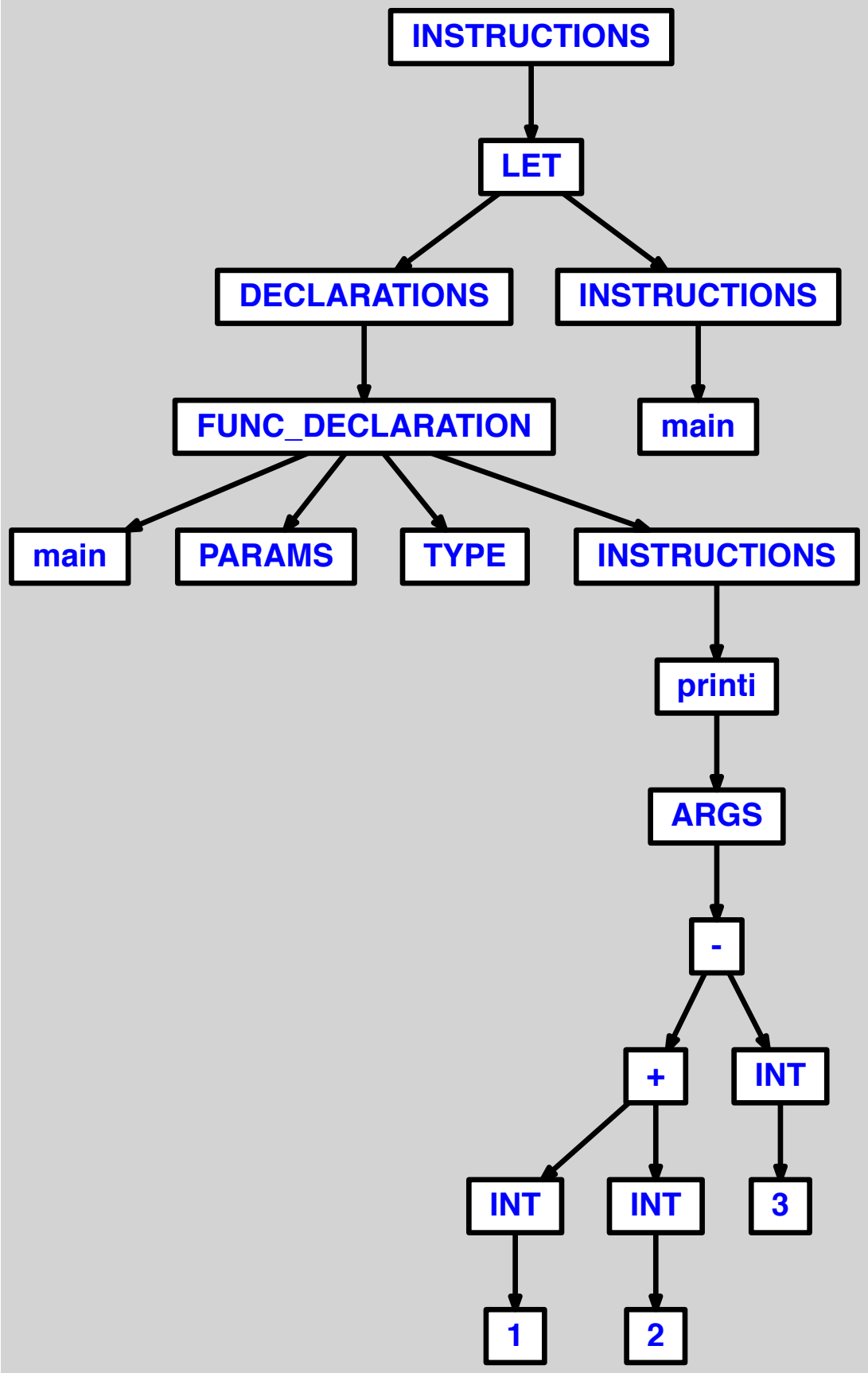
3.2.5 addition a 3 termes

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


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

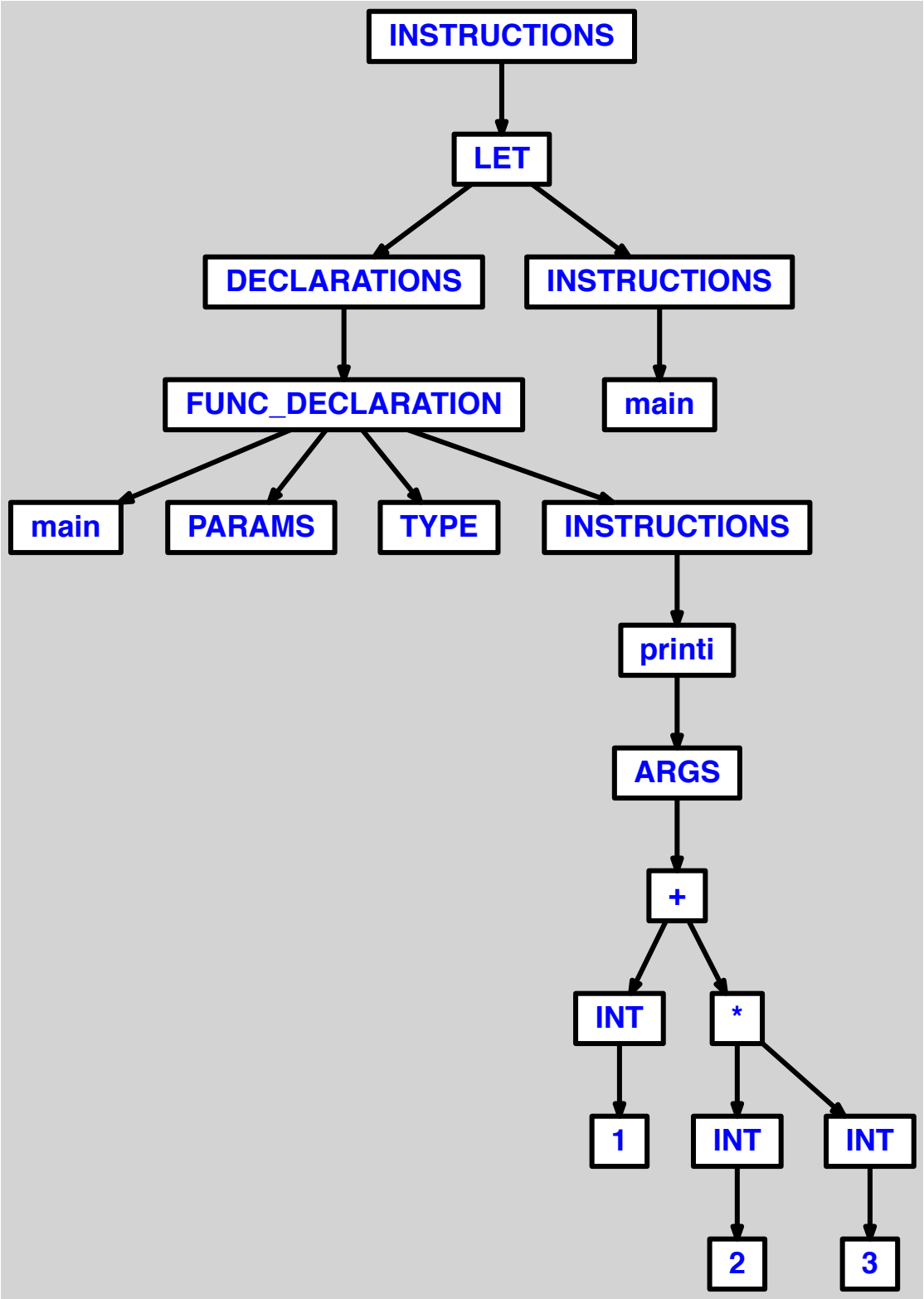
3.2.6 addition suivie de soustraction

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


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

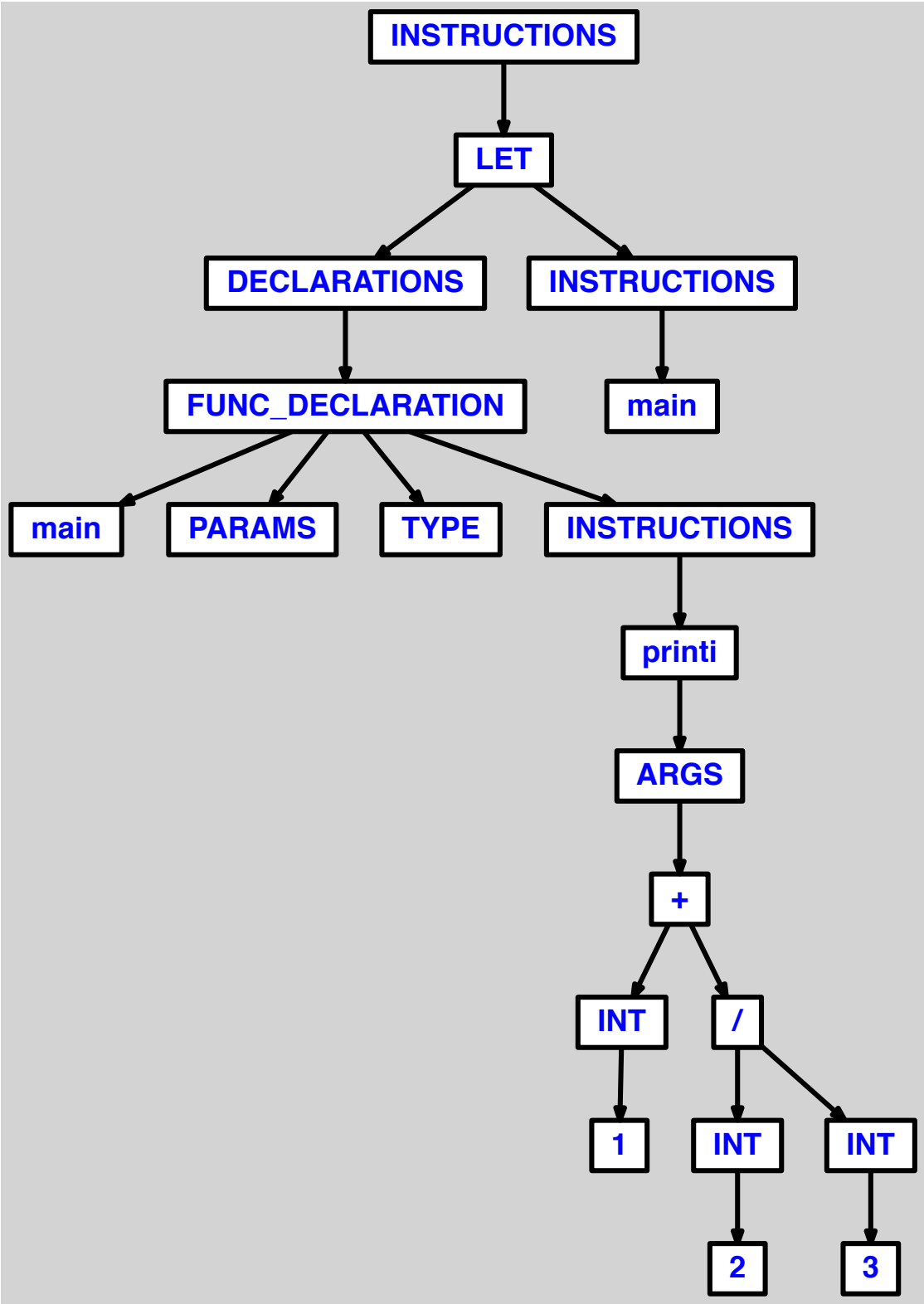
3.2.7 addition suivie de multiplication

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


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

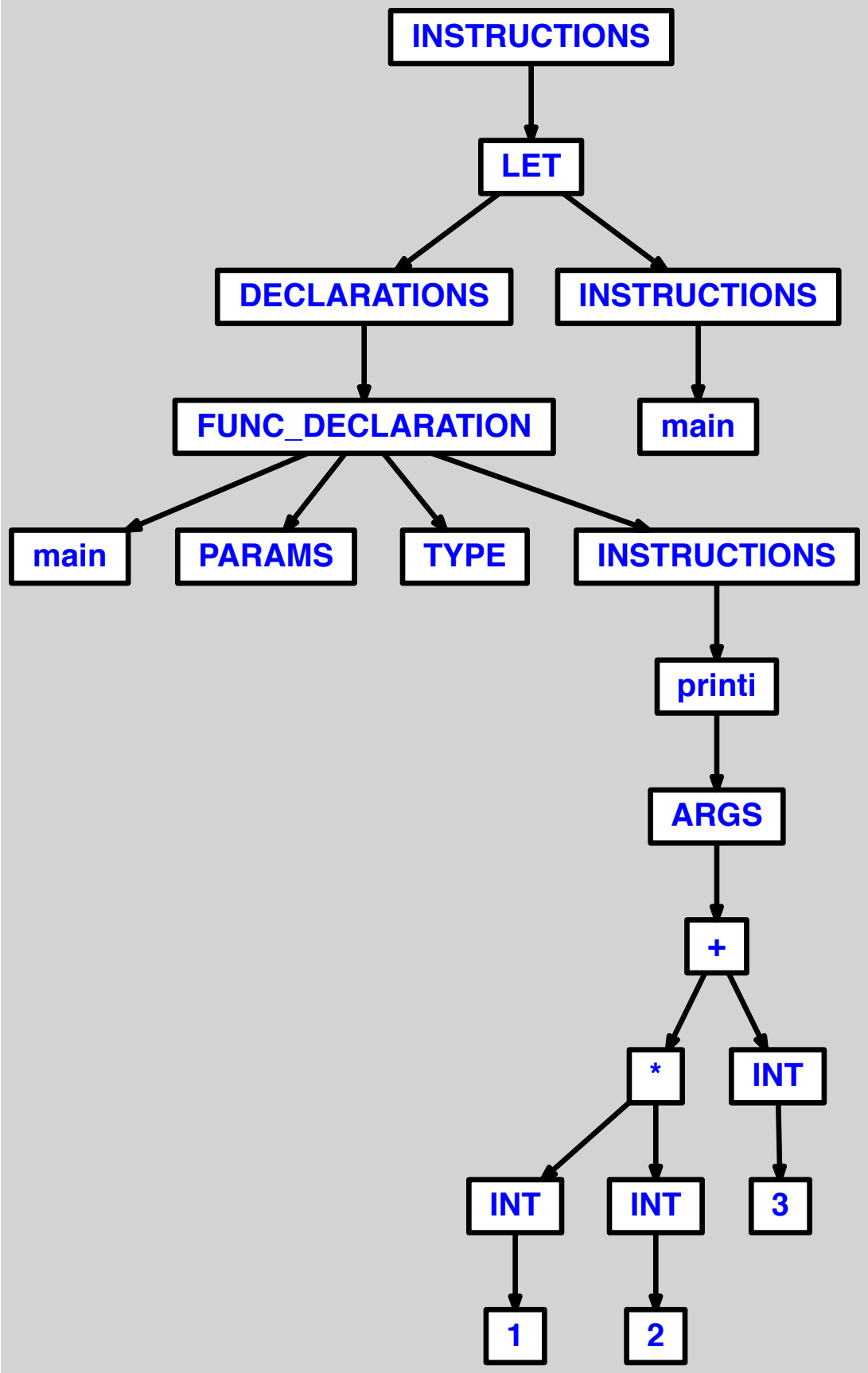
3.2.8 addition suivie de division

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


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

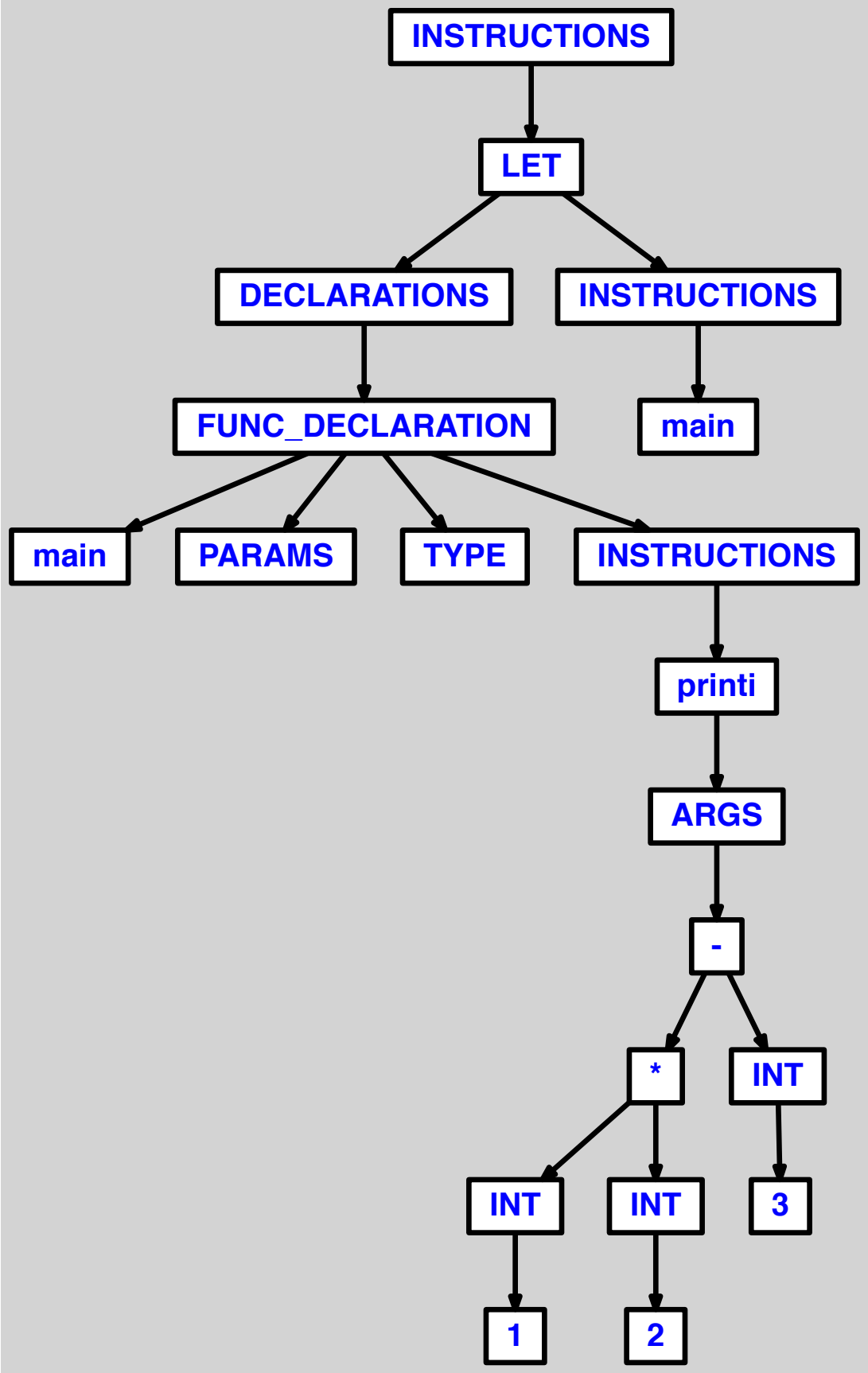
3.2.9 multiplication suivie d'addition

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


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

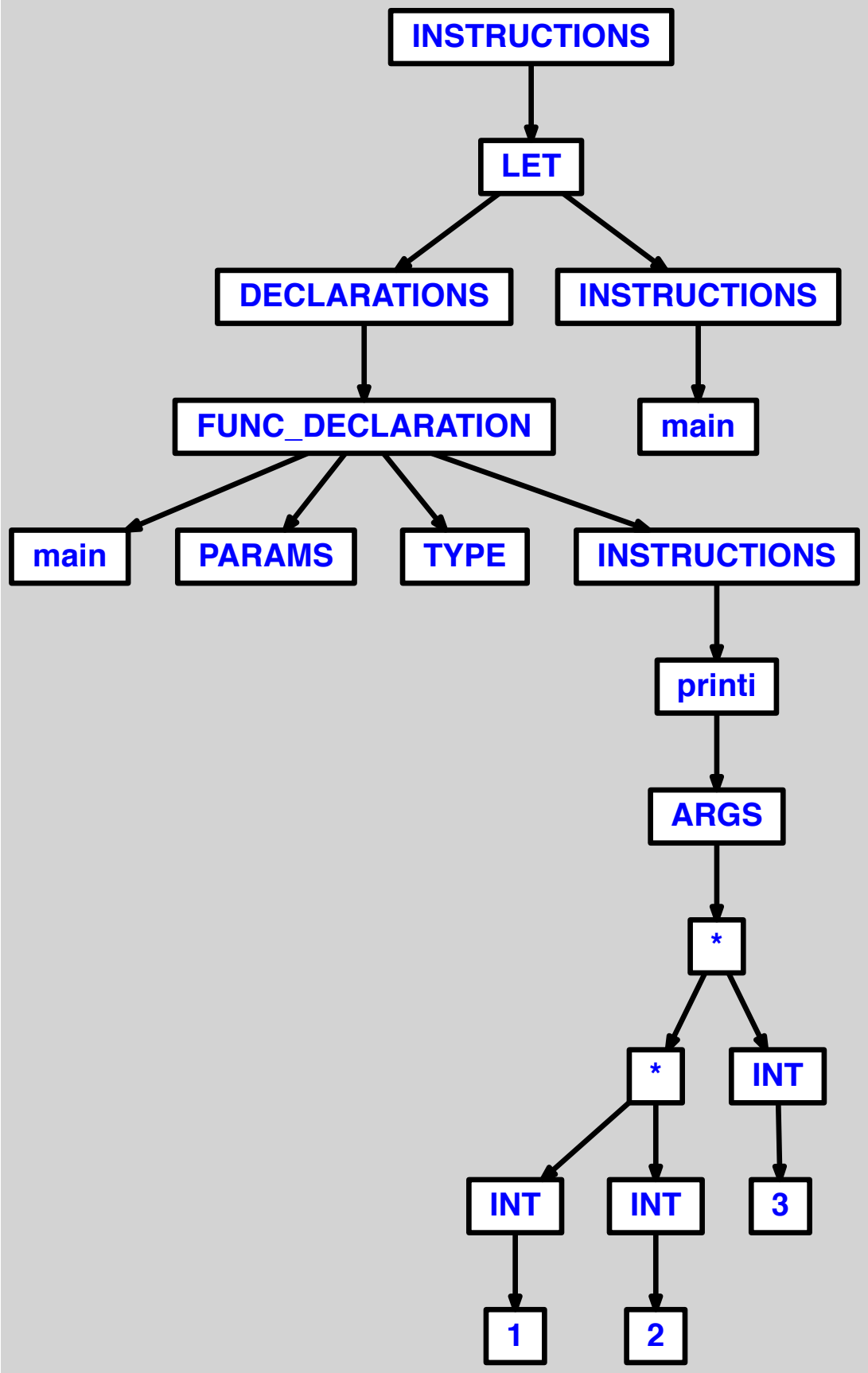
3.2.10 multiplication suivie de soustraction

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


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

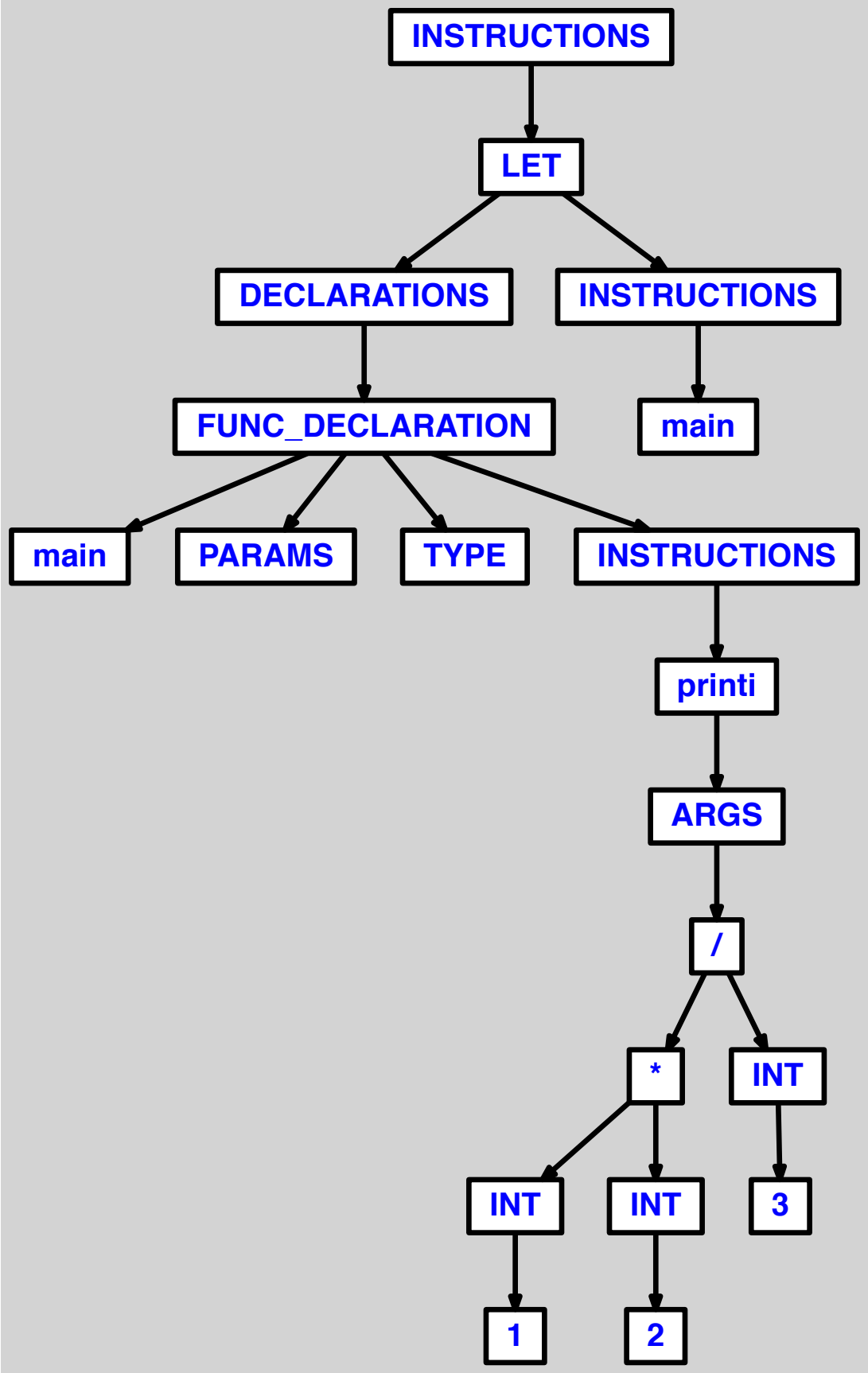
3.2.11 multiplication a 3 termes

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


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

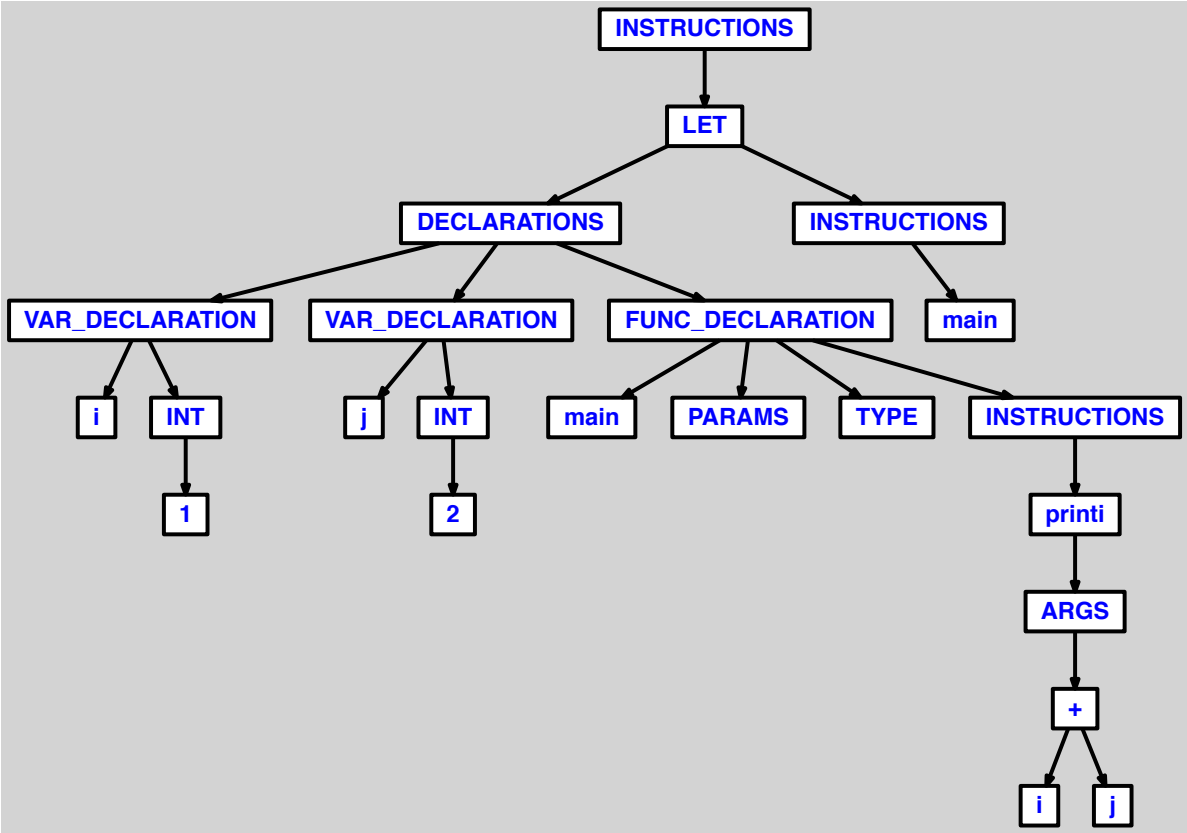
3.2.12 multiplication suivie de division

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


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

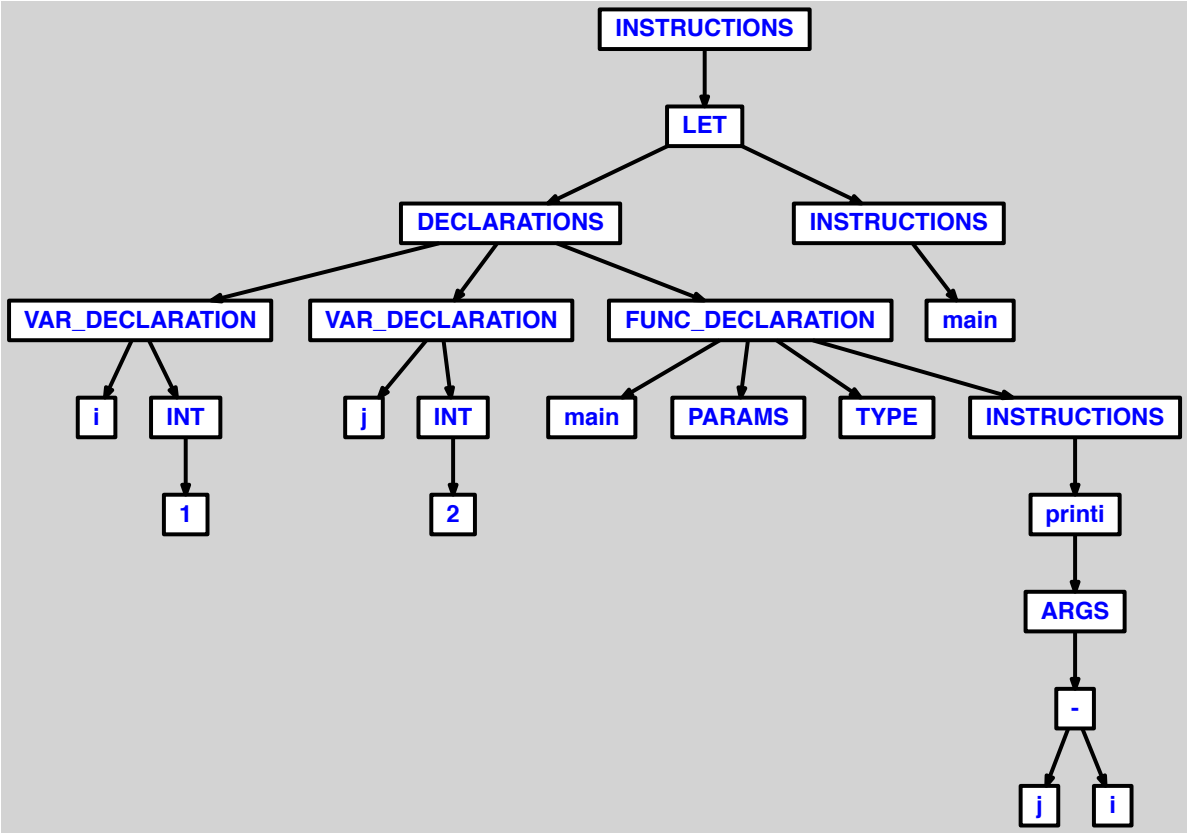
3.2.13 addition simple, a 2 termes, identifies par des variables

```
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
main
i
INT
j
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
printi
ARGS
+
i
j

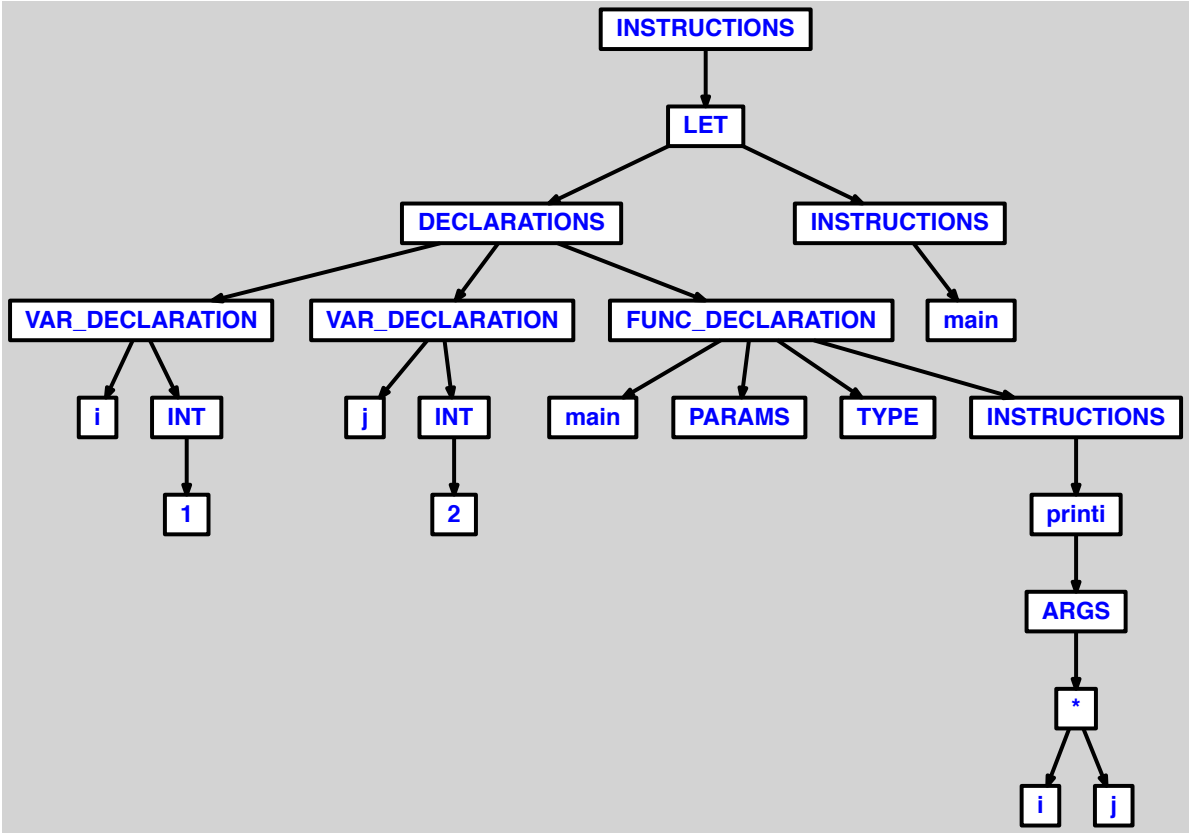
3.2.14 soustraction simple, a 2 termes, identifies par des variables

```
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
main
i
INT
j
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
printi
ARGS
-
j
i

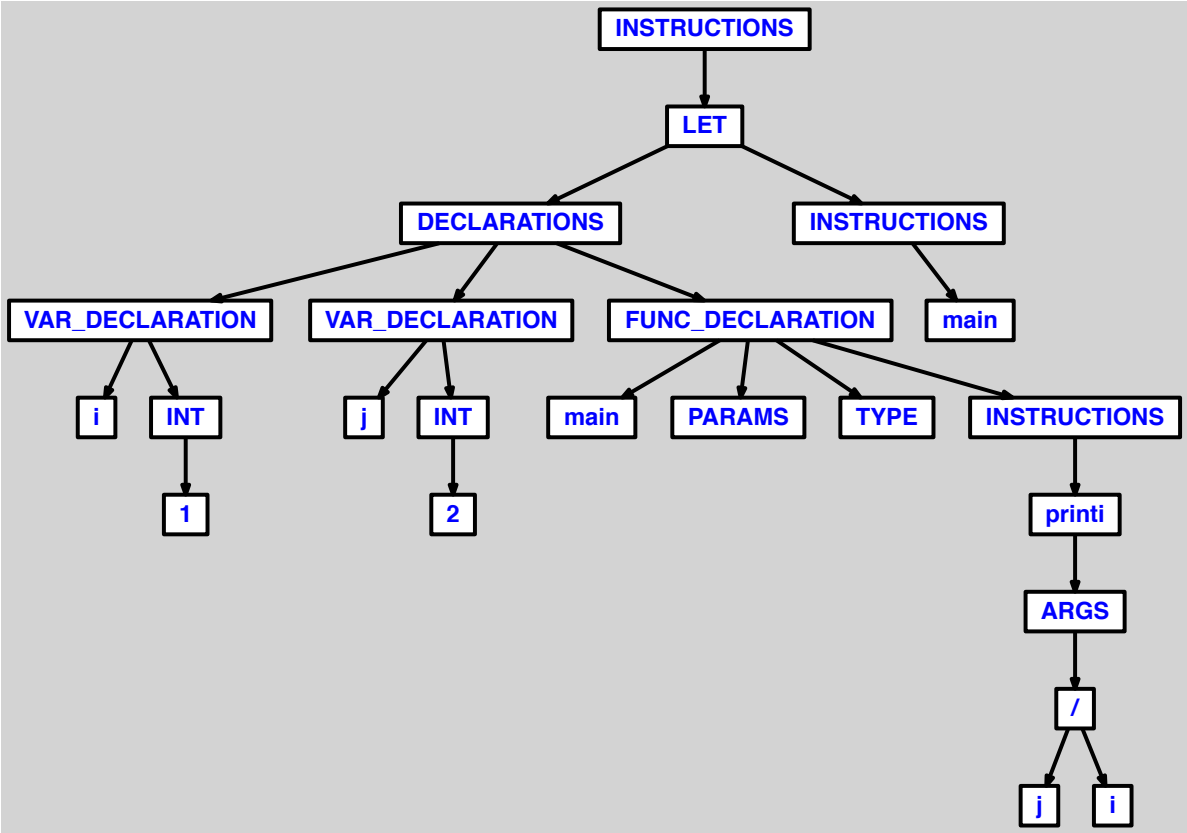
3.2.15 multiplication simple, a 2 termes, identifies par des variables

```
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
main
i
INT
j
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
printi
ARGS
*
i
j

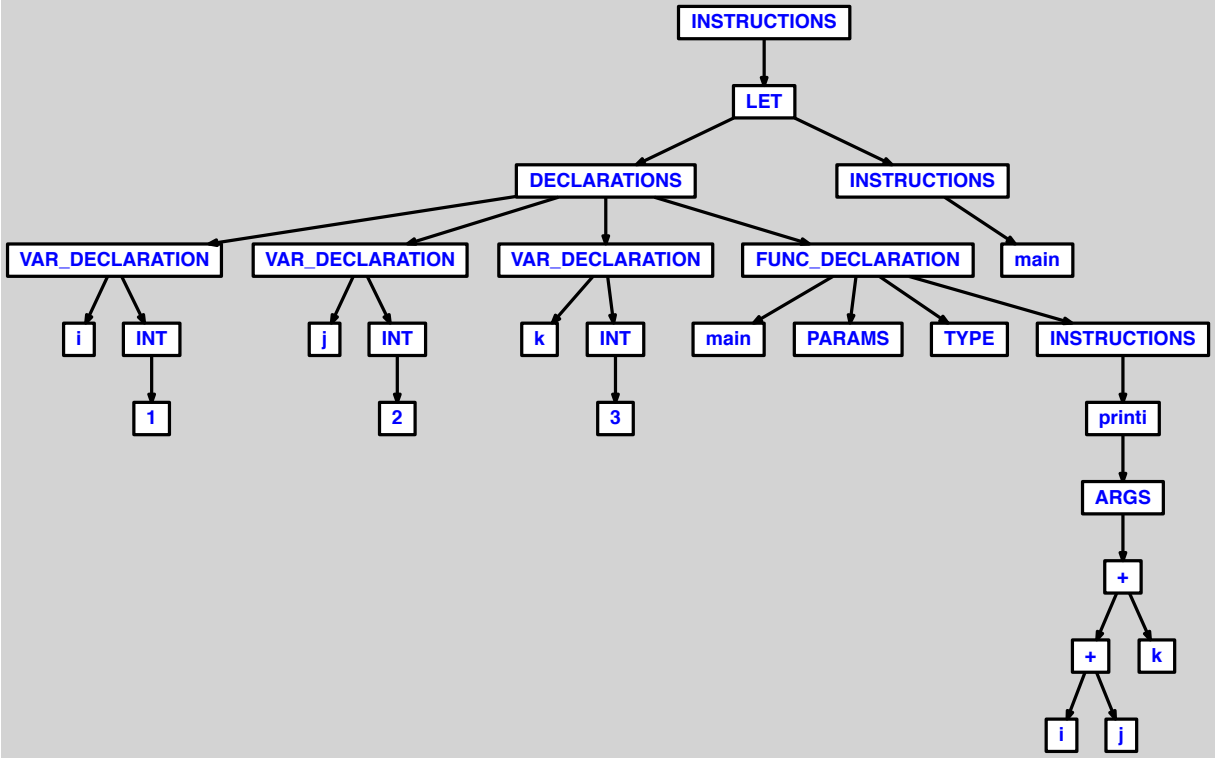
3.2.16 division simple, a 2 termes, identifies par des variables

```
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
main
i
INT
j
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
printi
ARGS
/
j
i

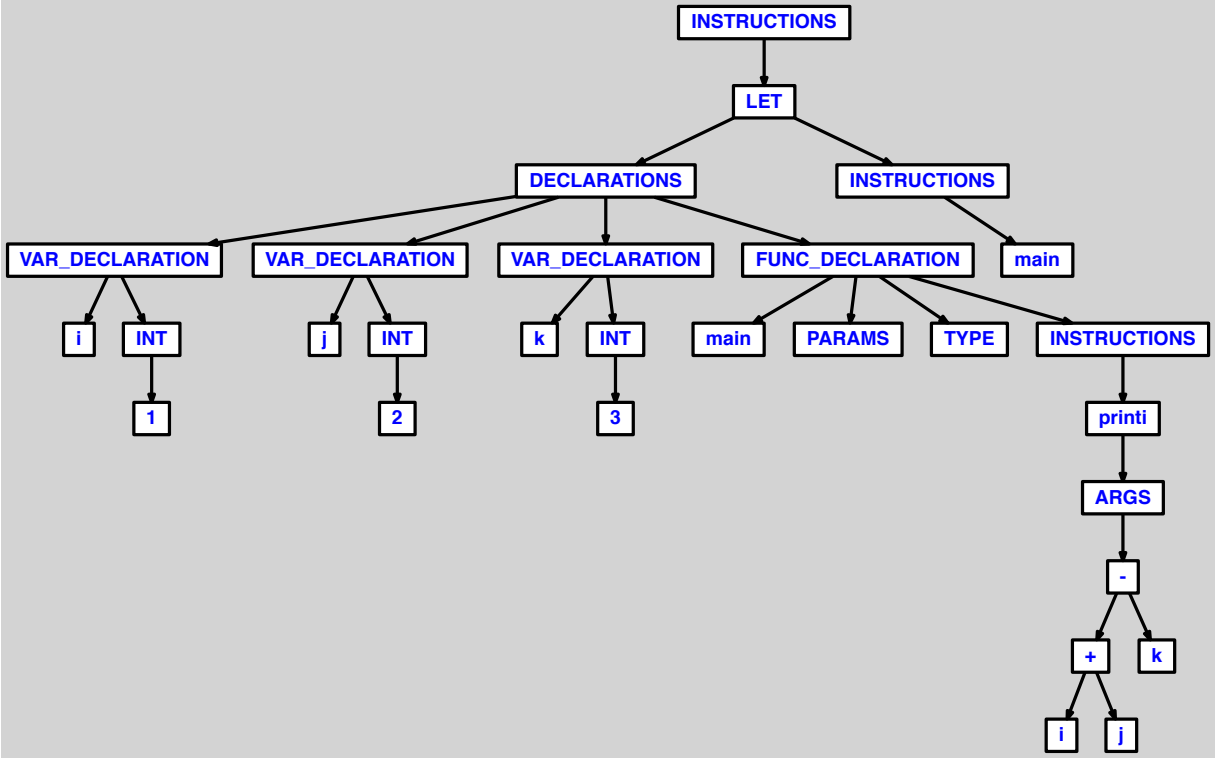
3.2.17 addition a 3 termes, identifies par des variables

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
+
k
i
j

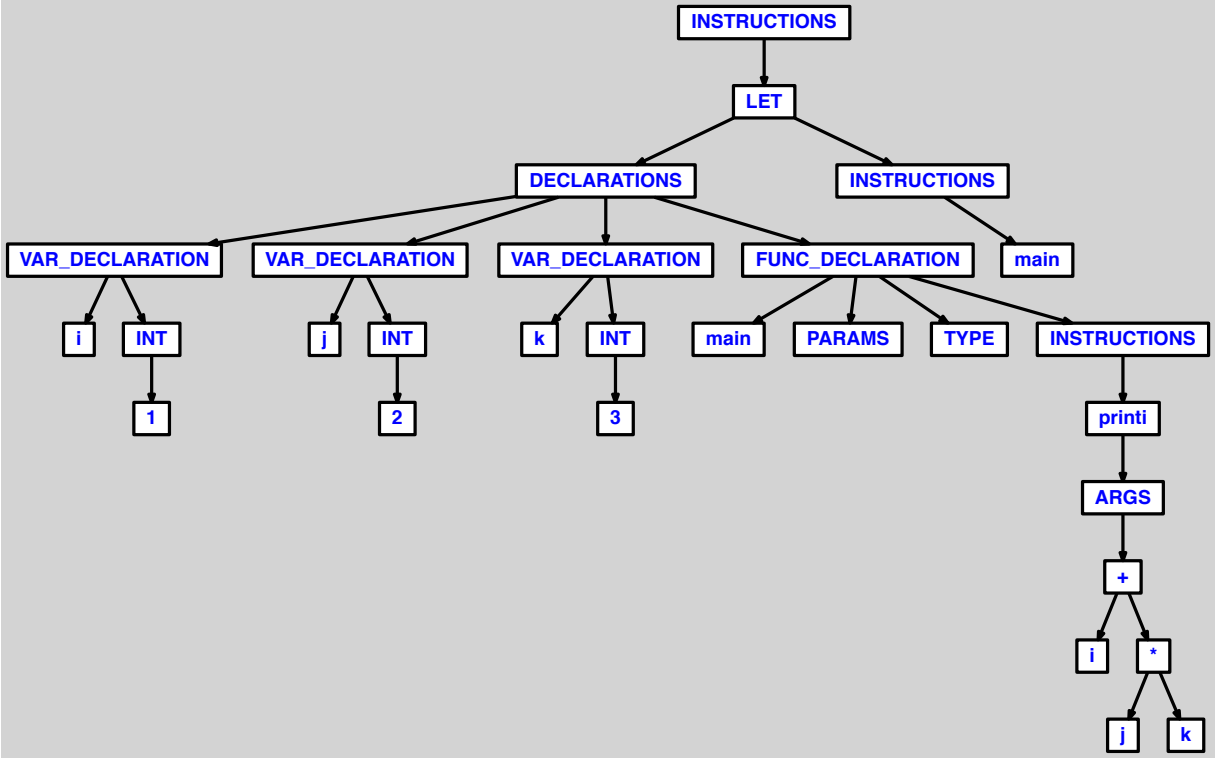
3.2.18 addition suivie de soustraction, avec termes identifies par variables

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
-
+
k
i
j

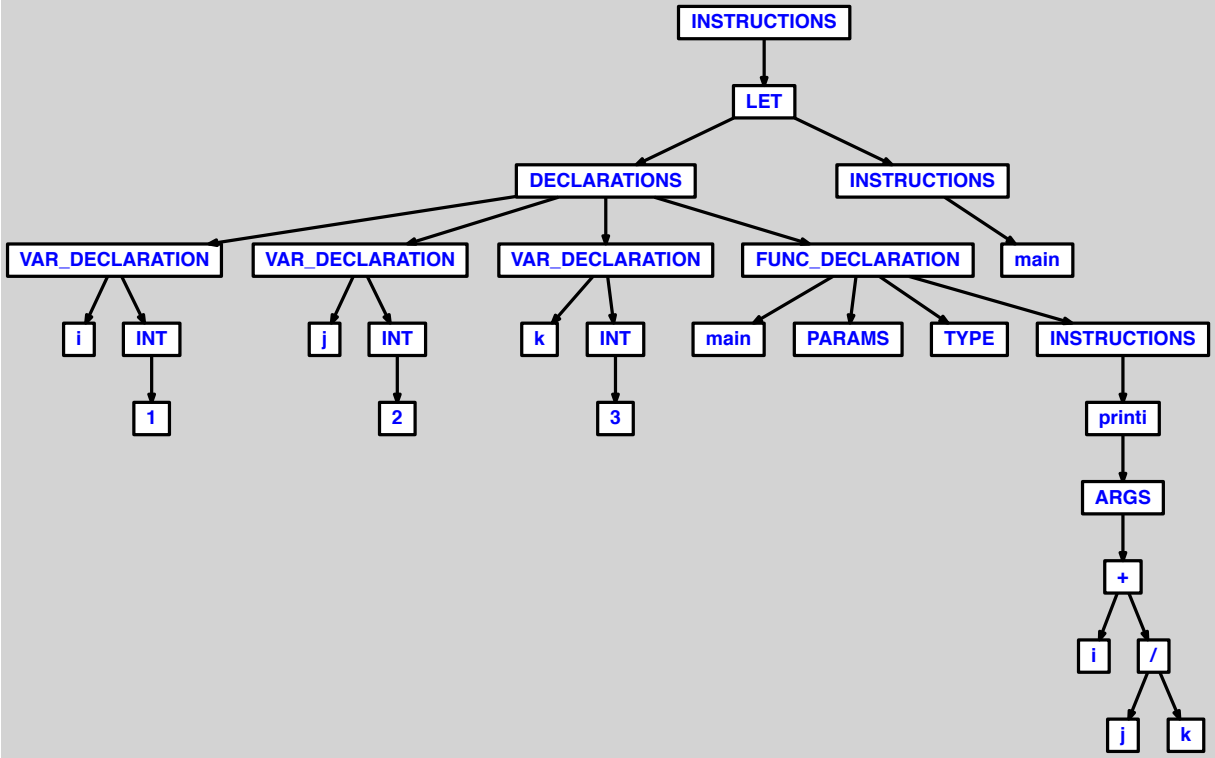
3.2.19 addition suivie de multiplication, avec termes identifies par variables

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
i
*
j
k

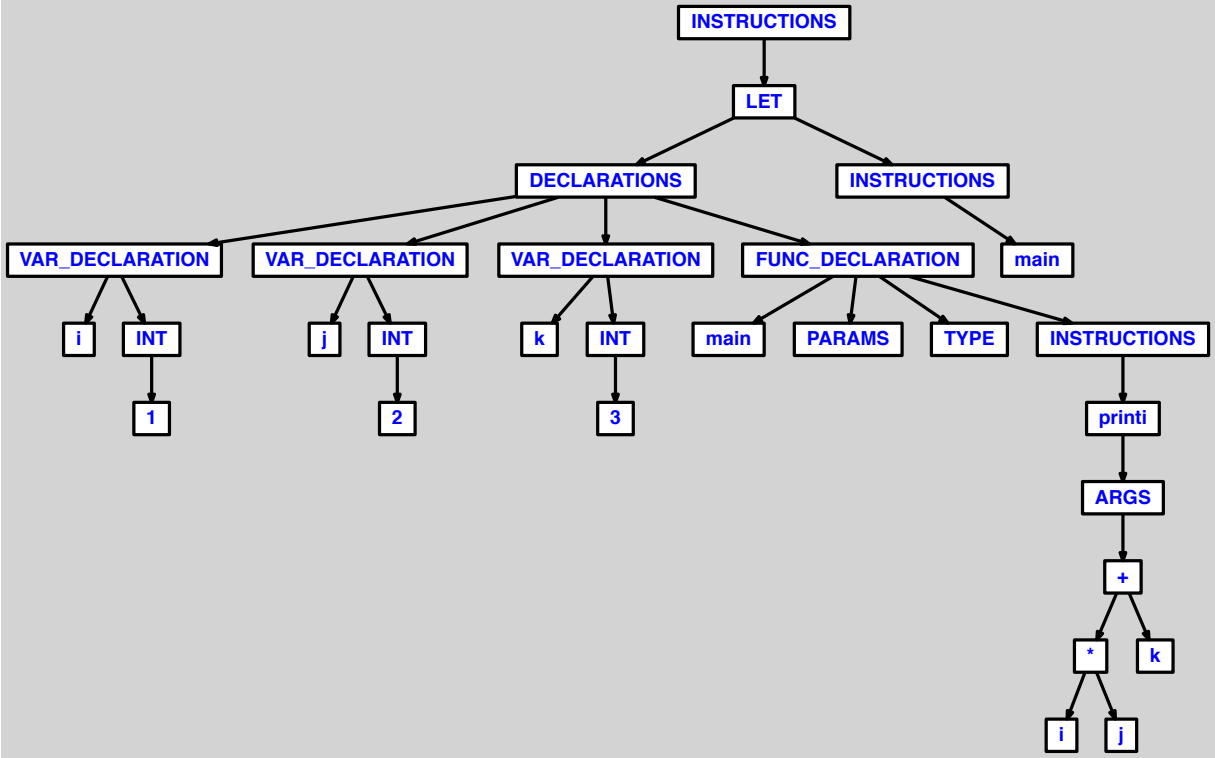
3.2.20 addition suivie de division, avec termes identifies par variables

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
i
/
j
k

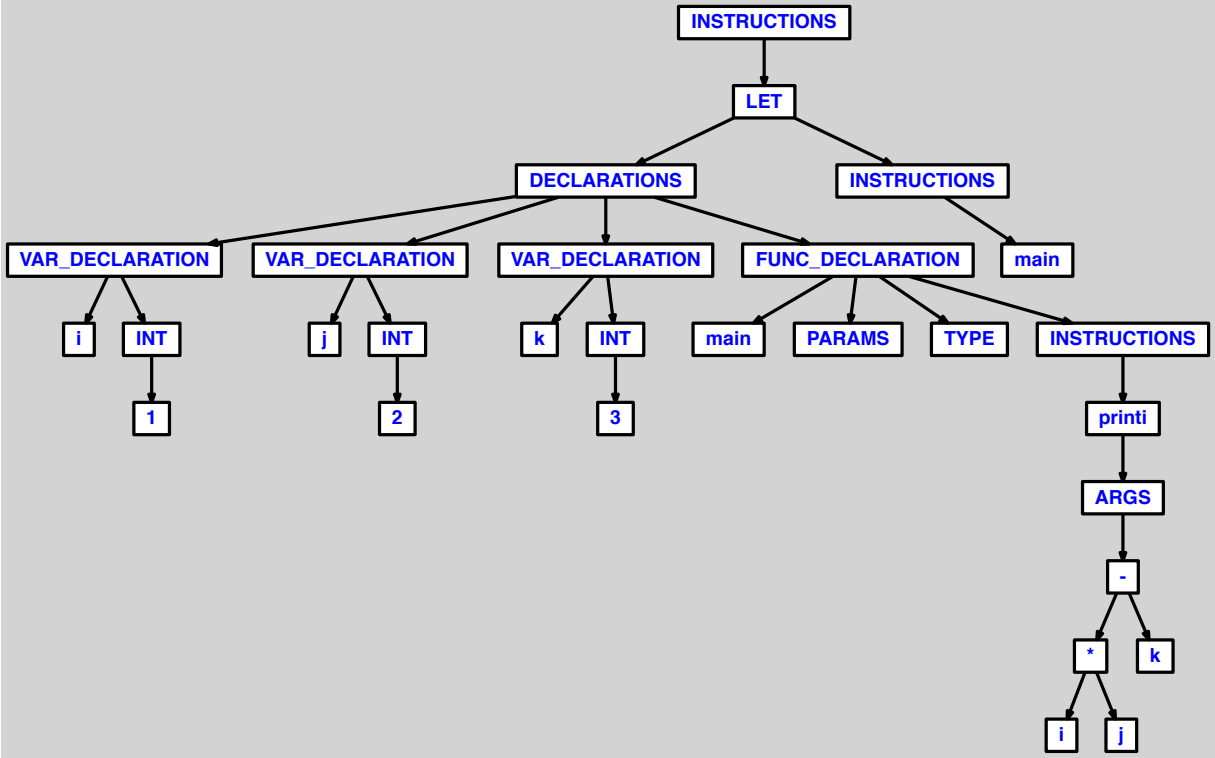
3.2.21 multiplication suivie d'addition, avec termes identifies par variables

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
*
k
i
j

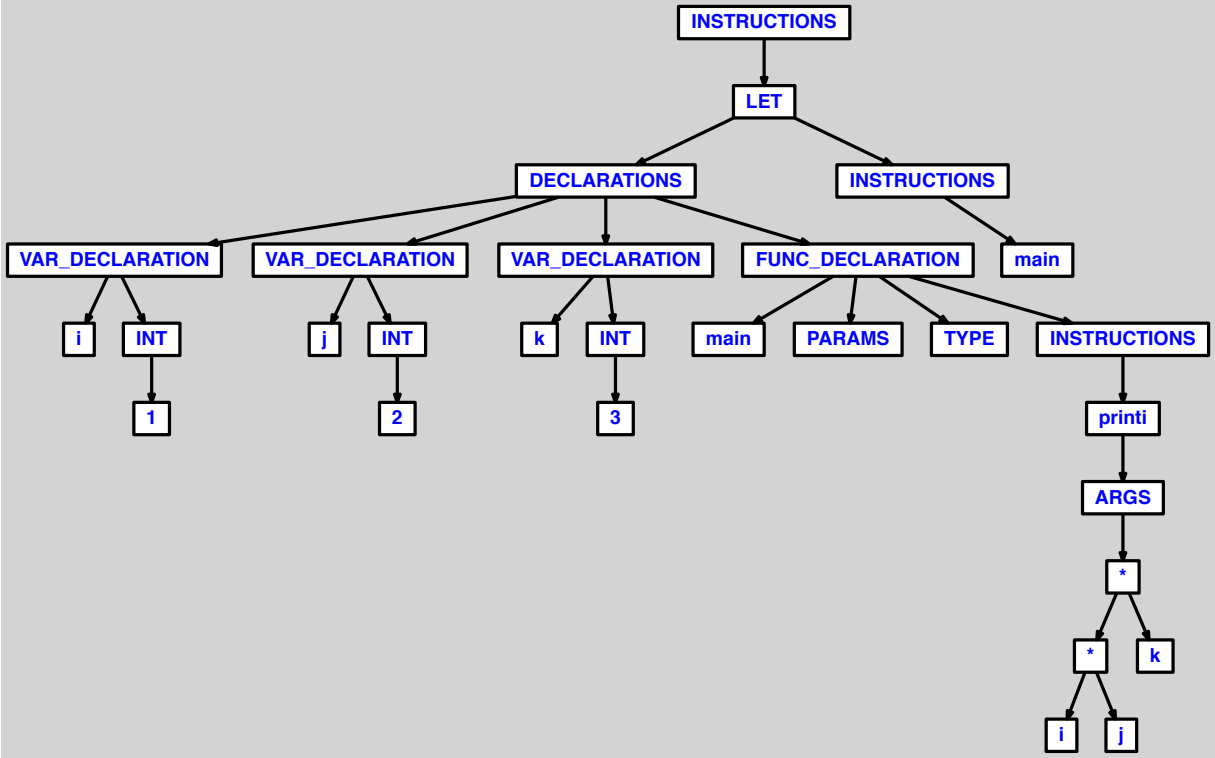
3.2.22 multiplication suivie de soustraction, avec termes identifies par variables

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
-
*
k
i
j

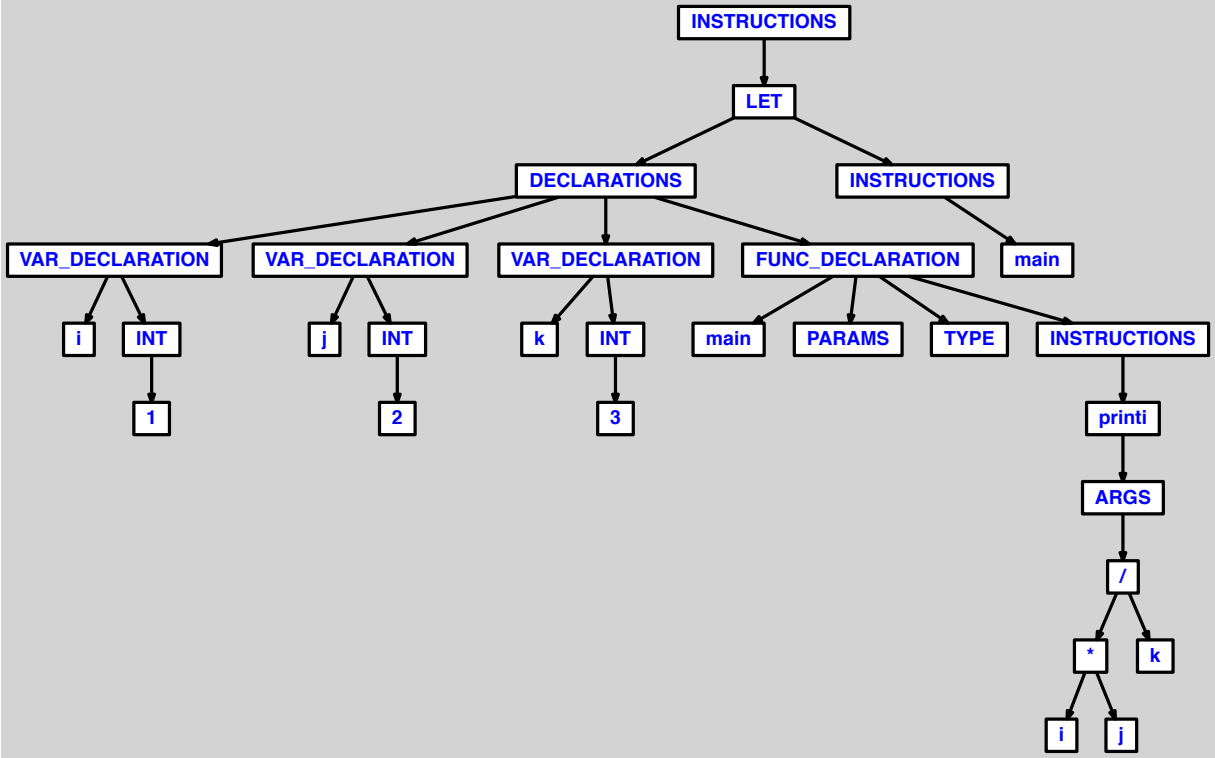
3.2.23 multiplication a 3 termes, identifies par des variables

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
*
*
k
i
j

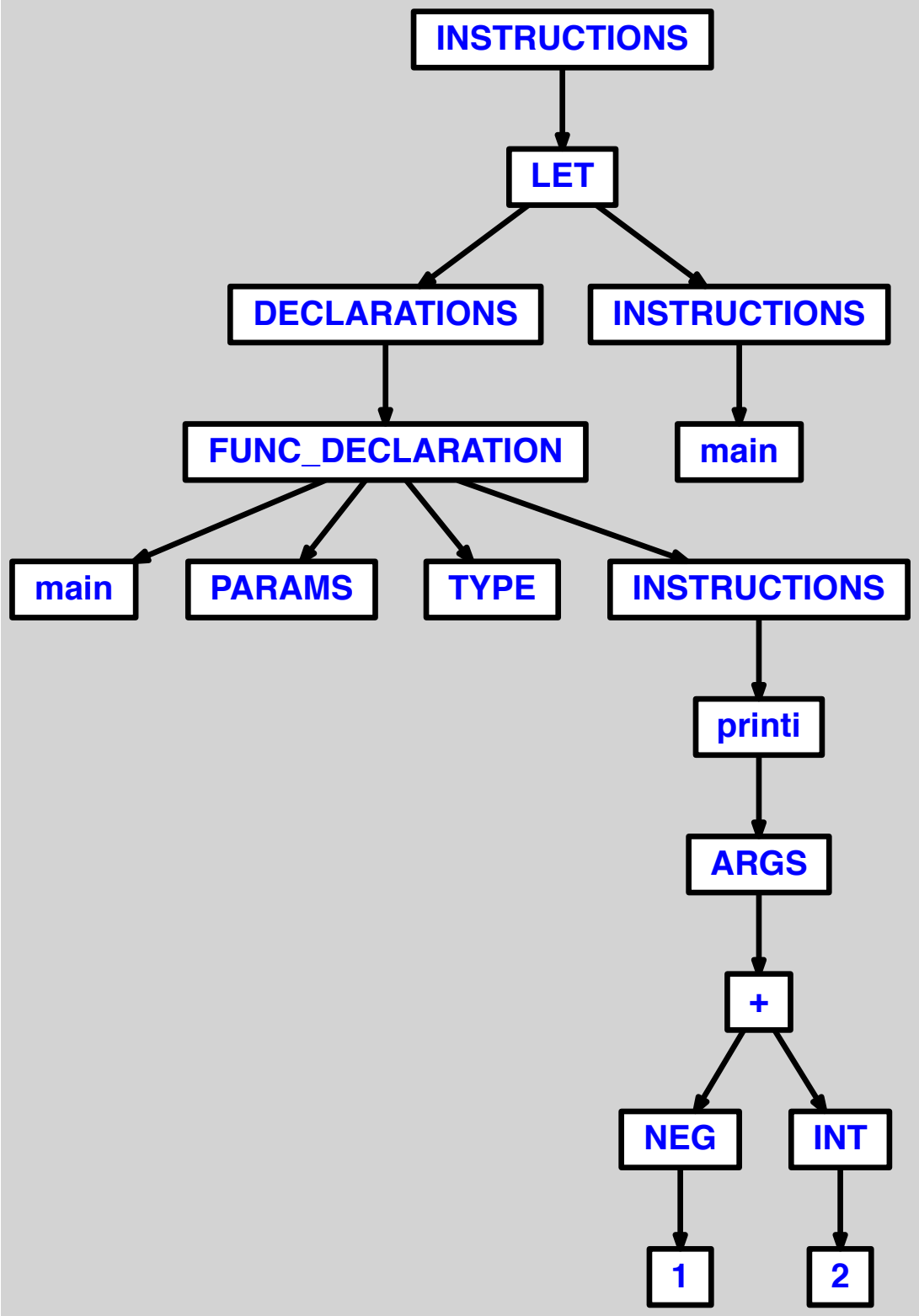
3.2.24 mutliplication suivie de division, avec termes identifies par variables

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
/
*
k
i
j

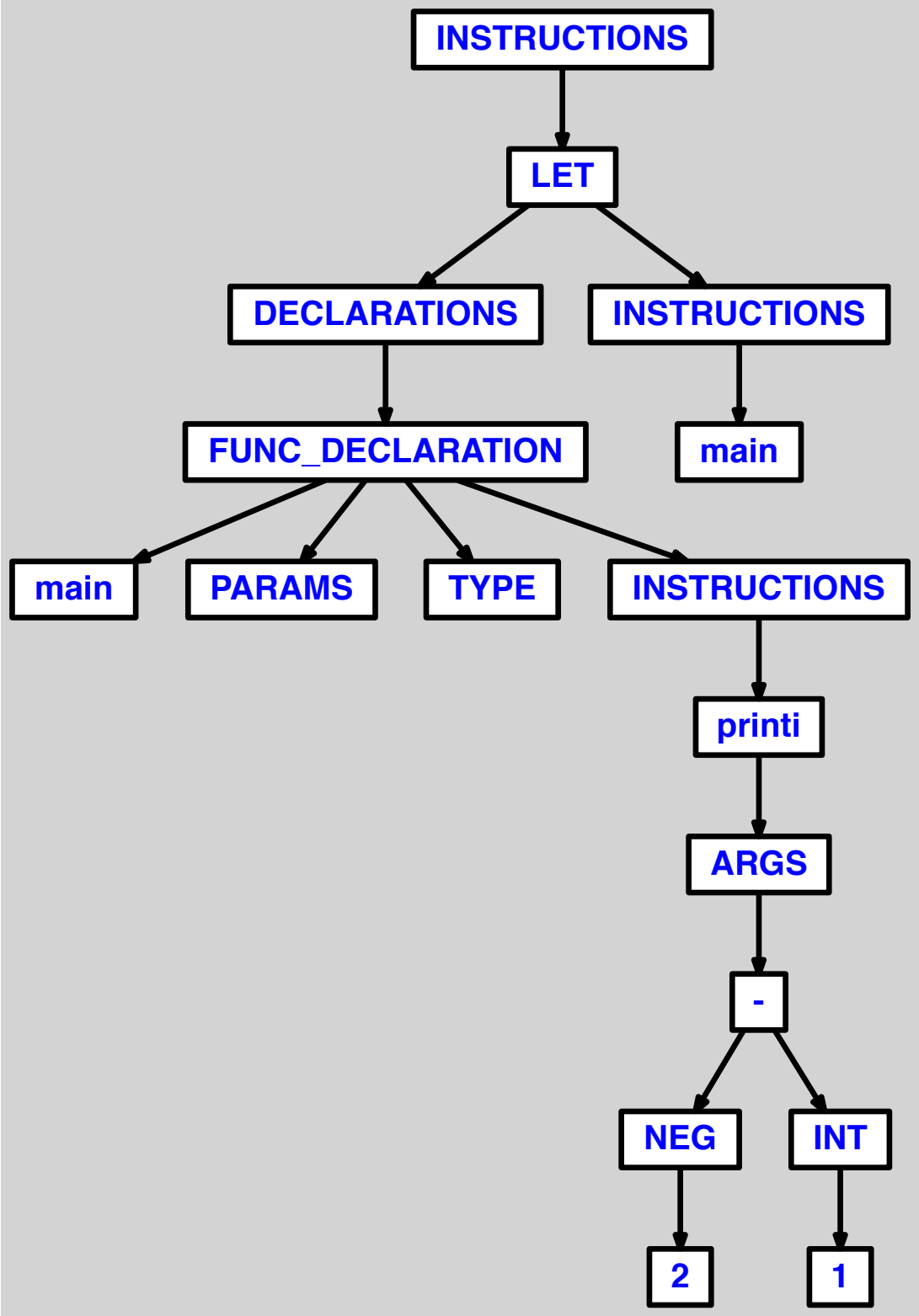
3.2.25 addition simple, a 2 termes, dont un ayant moins unaire

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


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

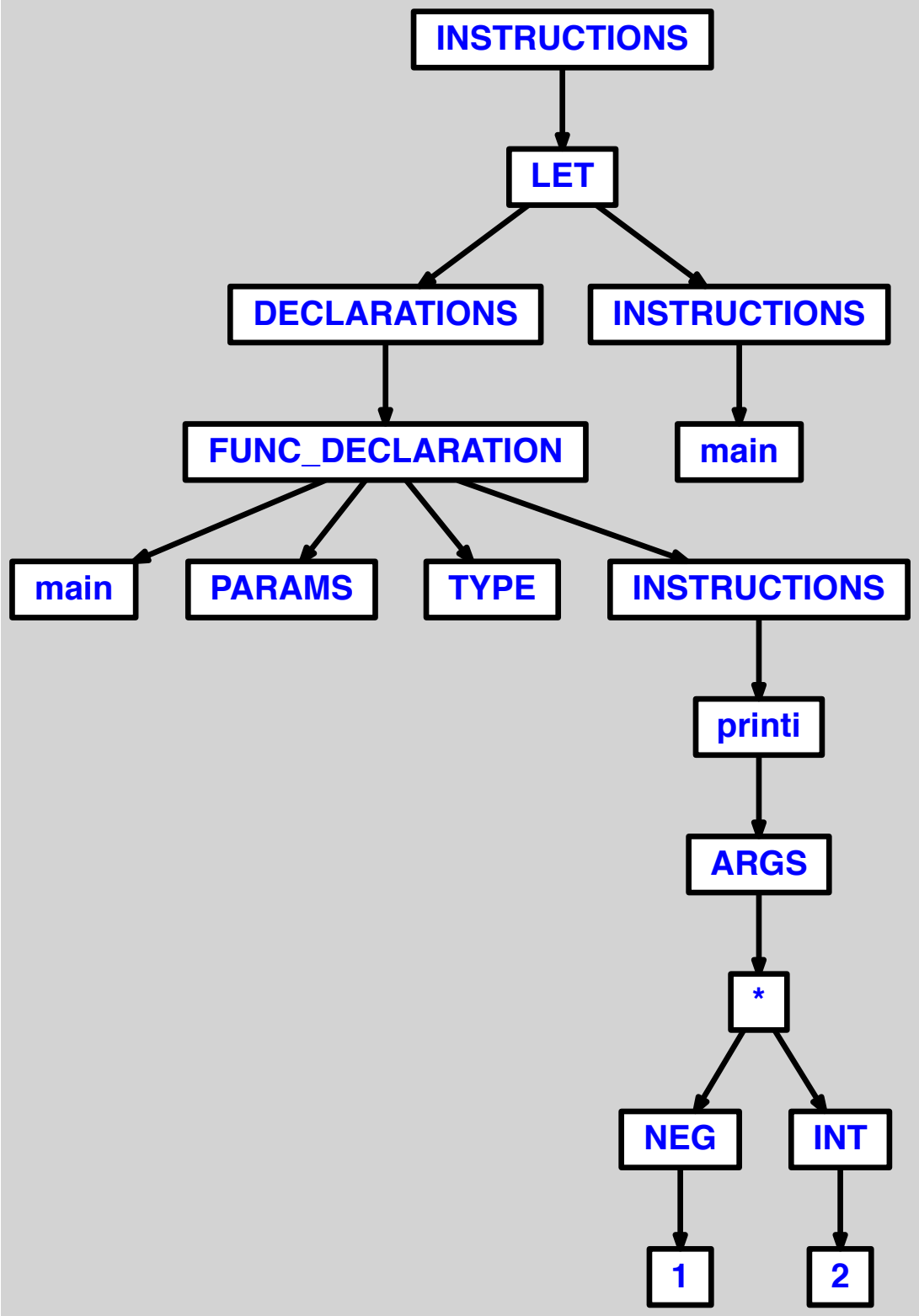
3.2.26 soustraction simple, a 2 termes, dont ayant moins unaire

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


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

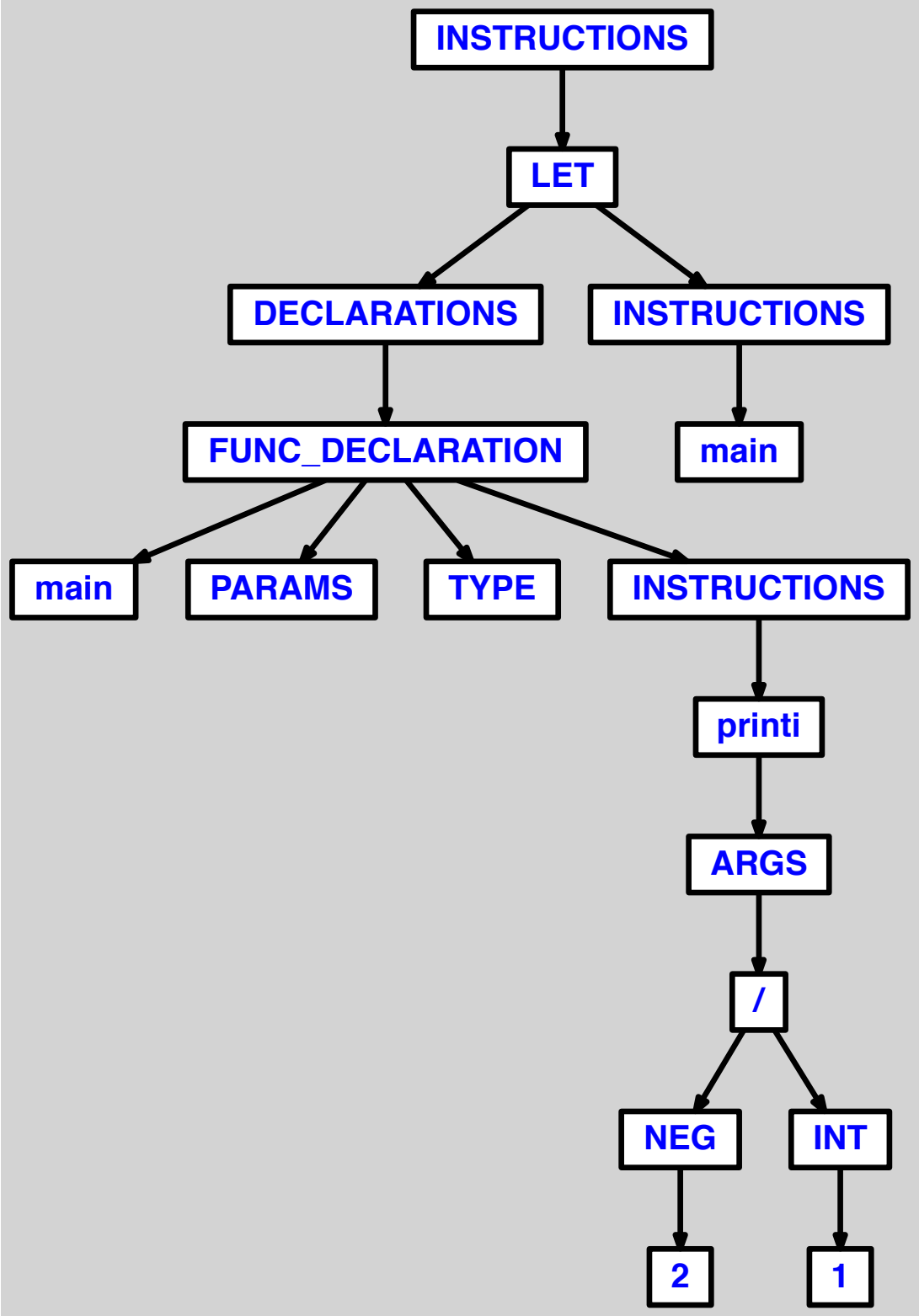
3.2.27 multiplication simple, a 2 termes, dont un ayant moins unaire

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


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

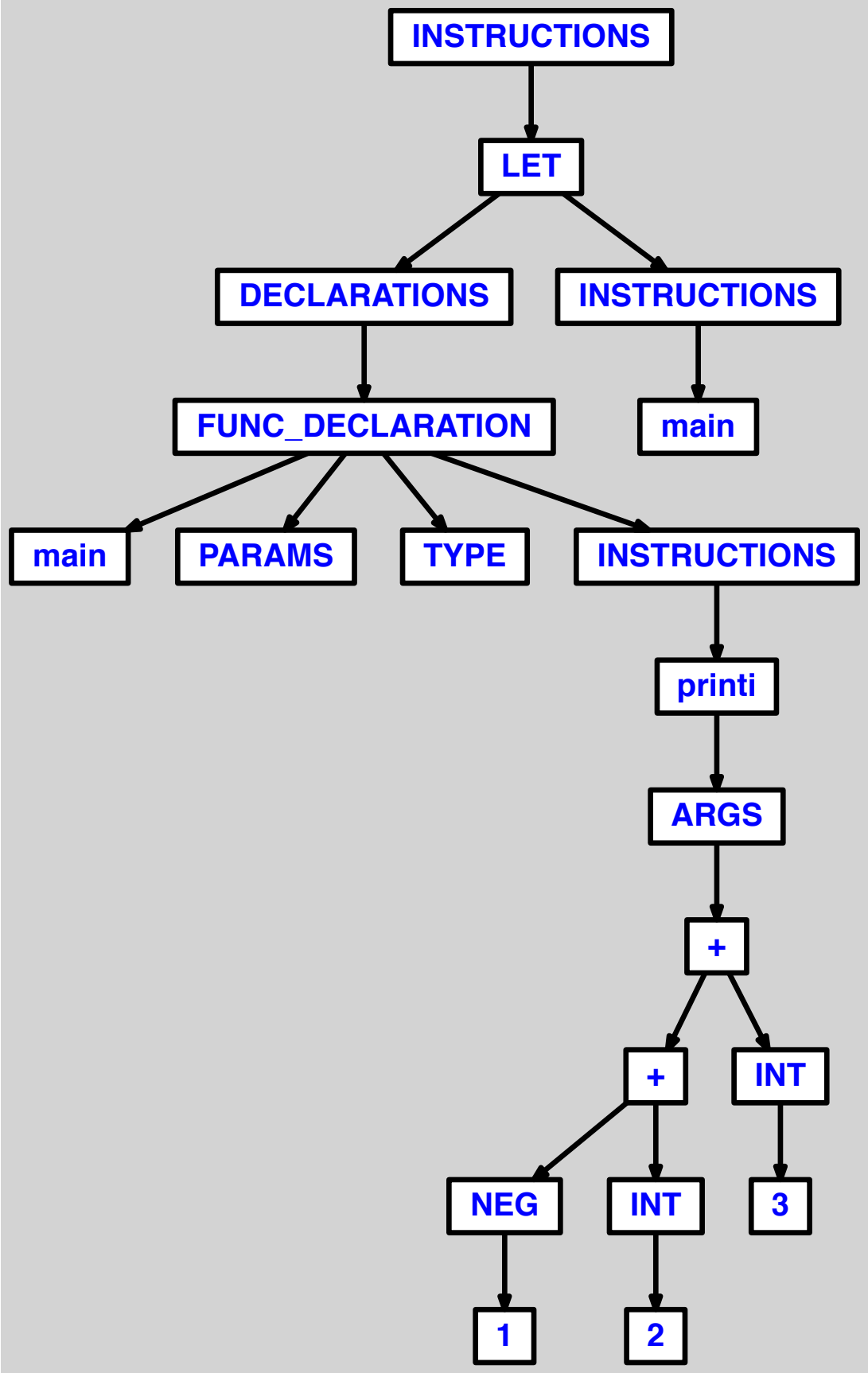
3.2.28 division simple, a 2 termes, dont un ayant moins unaire

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


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

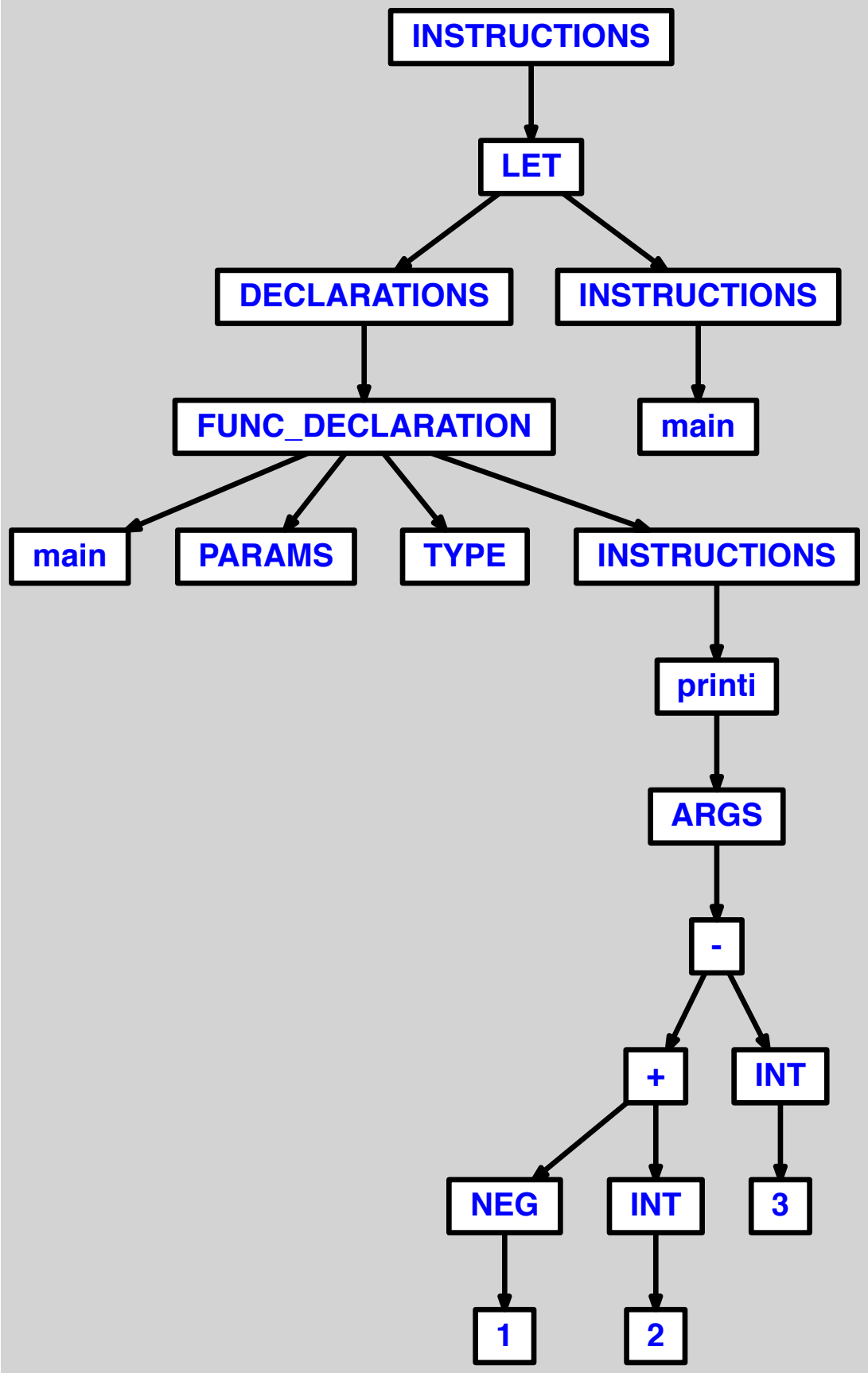
3.2.29 addition a 3 termes, dont un ayant moins unaire

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


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

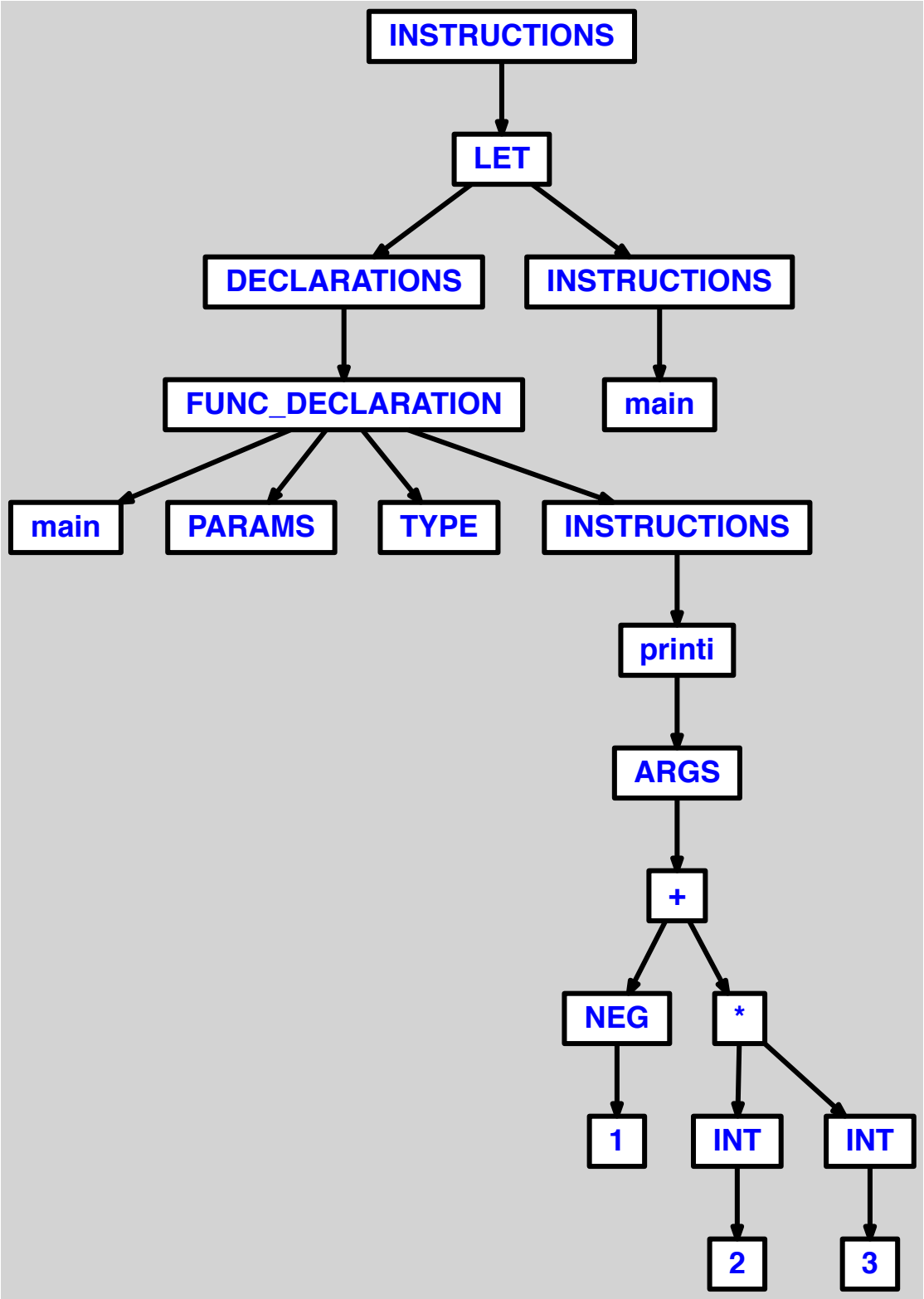
3.2.30 addition suivie de soustraction, avec un terme ayant moins unaire

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


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

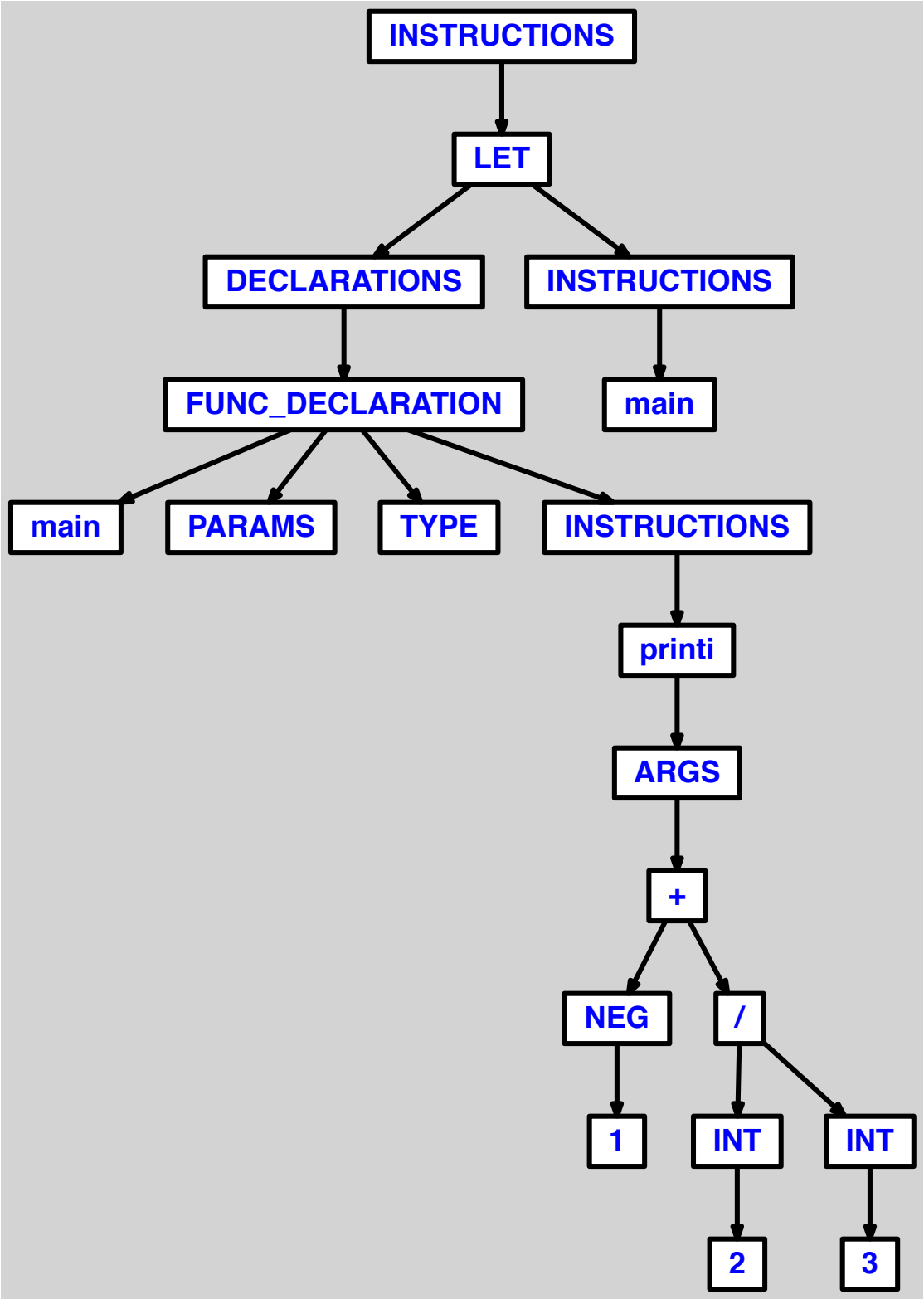
3.2.31 addition suivie de multiplication, avec un terme ayant moins unaire

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


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

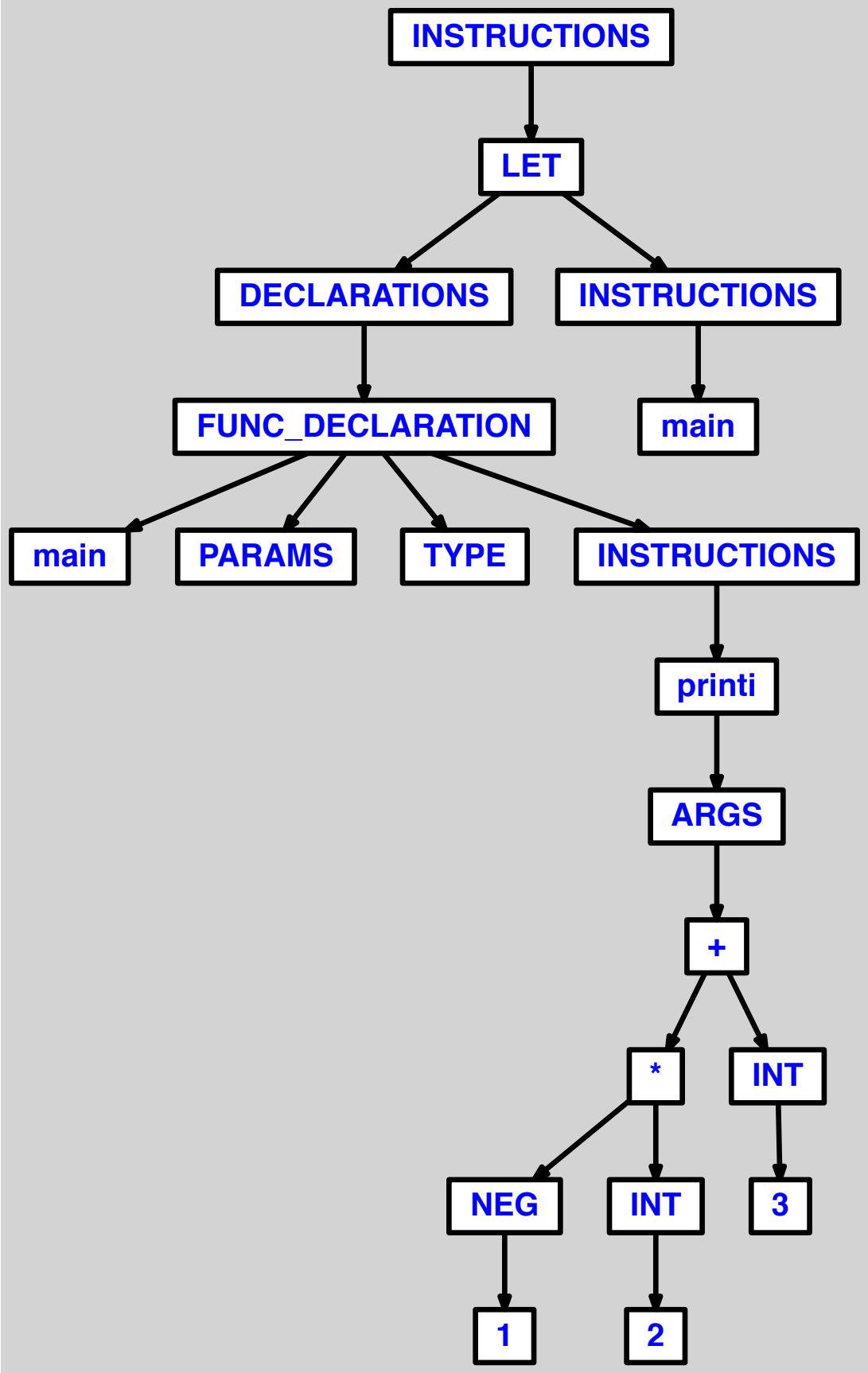
3.2.32 addition suivie de division, avec un terme ayant moins unaire

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


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

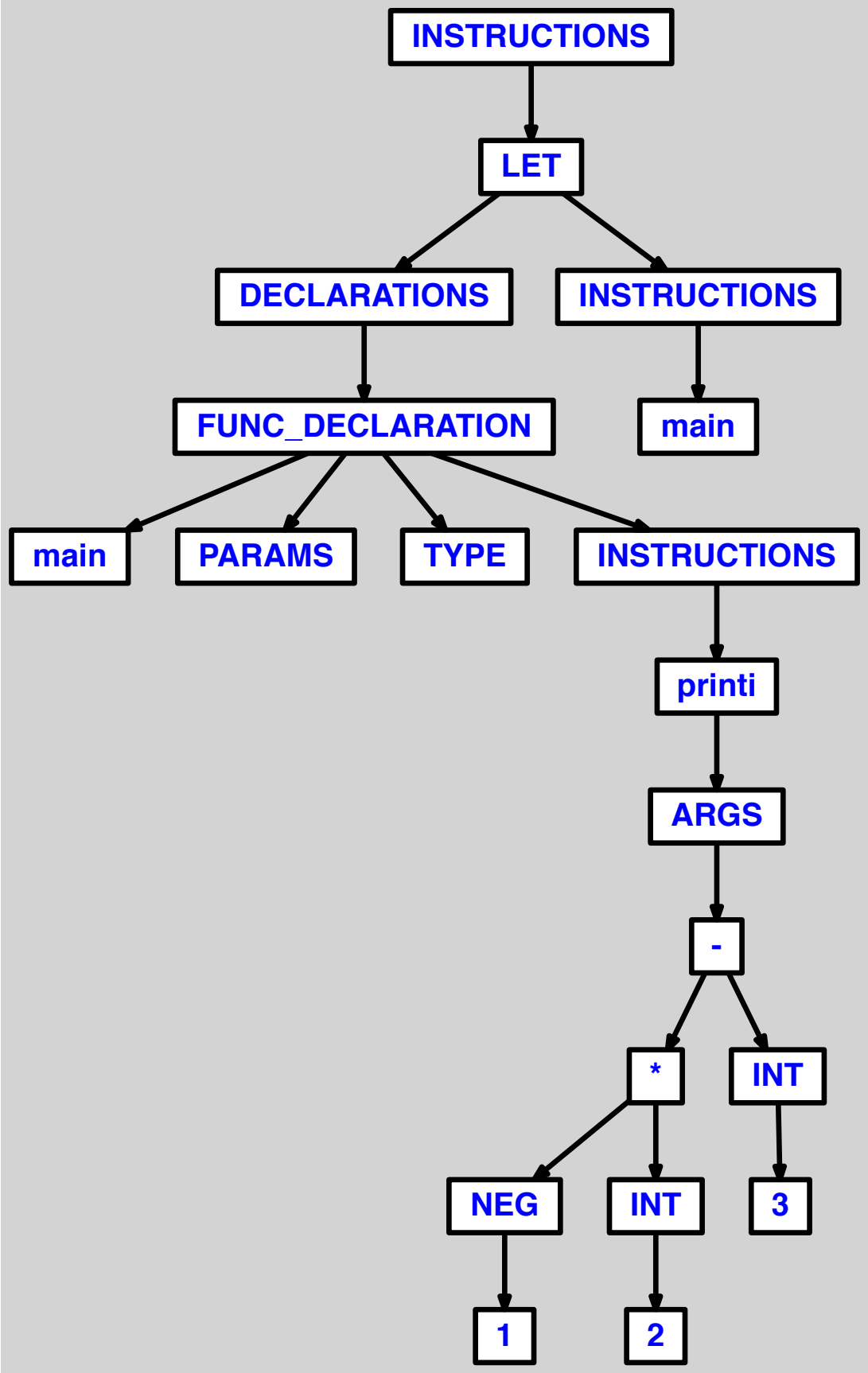
3.2.33 multiplication suivie d'addition, dont un terme ayant moins unaire

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


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

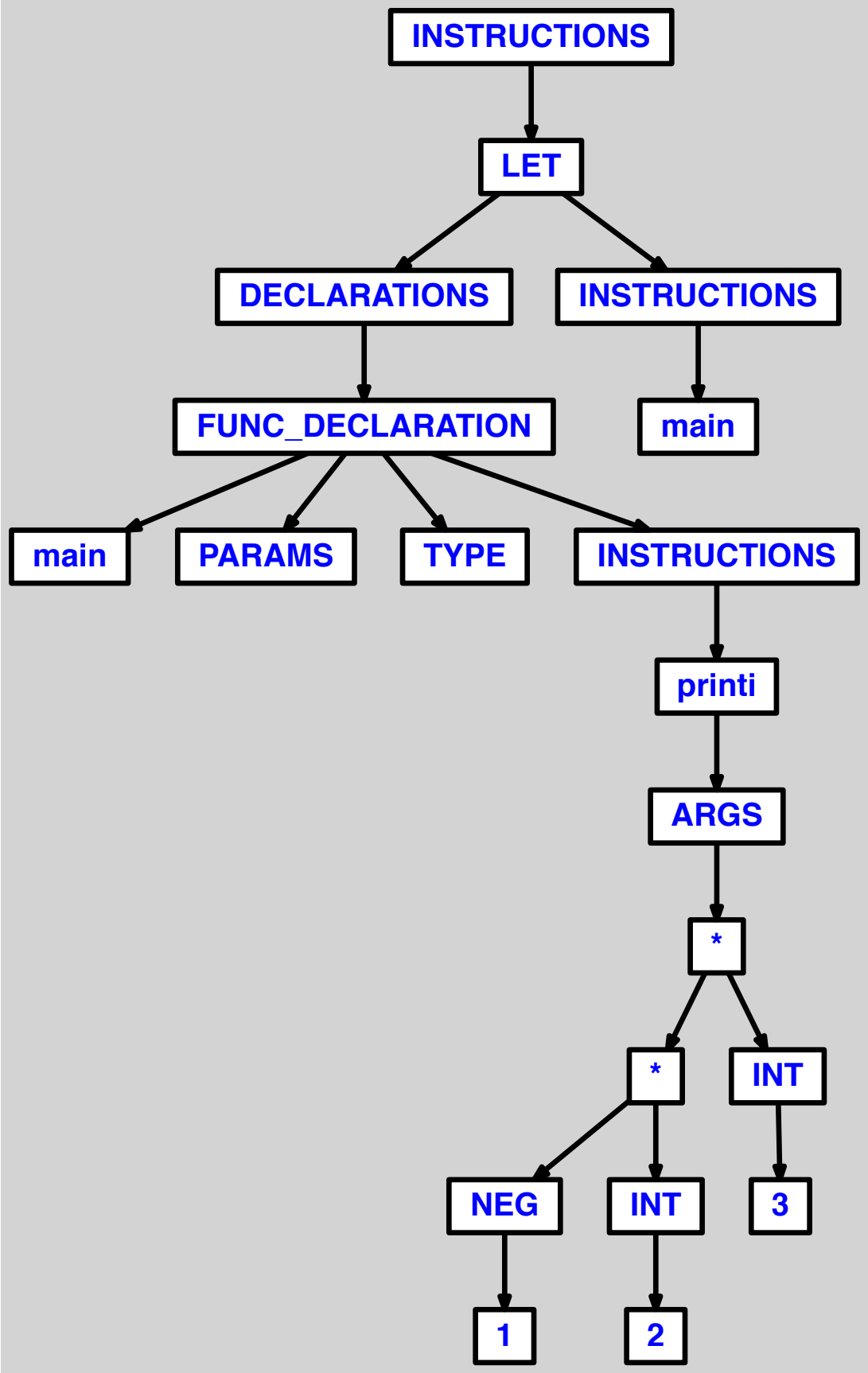
3.2.34 multiplication suivie de soustraction, dont un terme ayant moins unaire

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


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

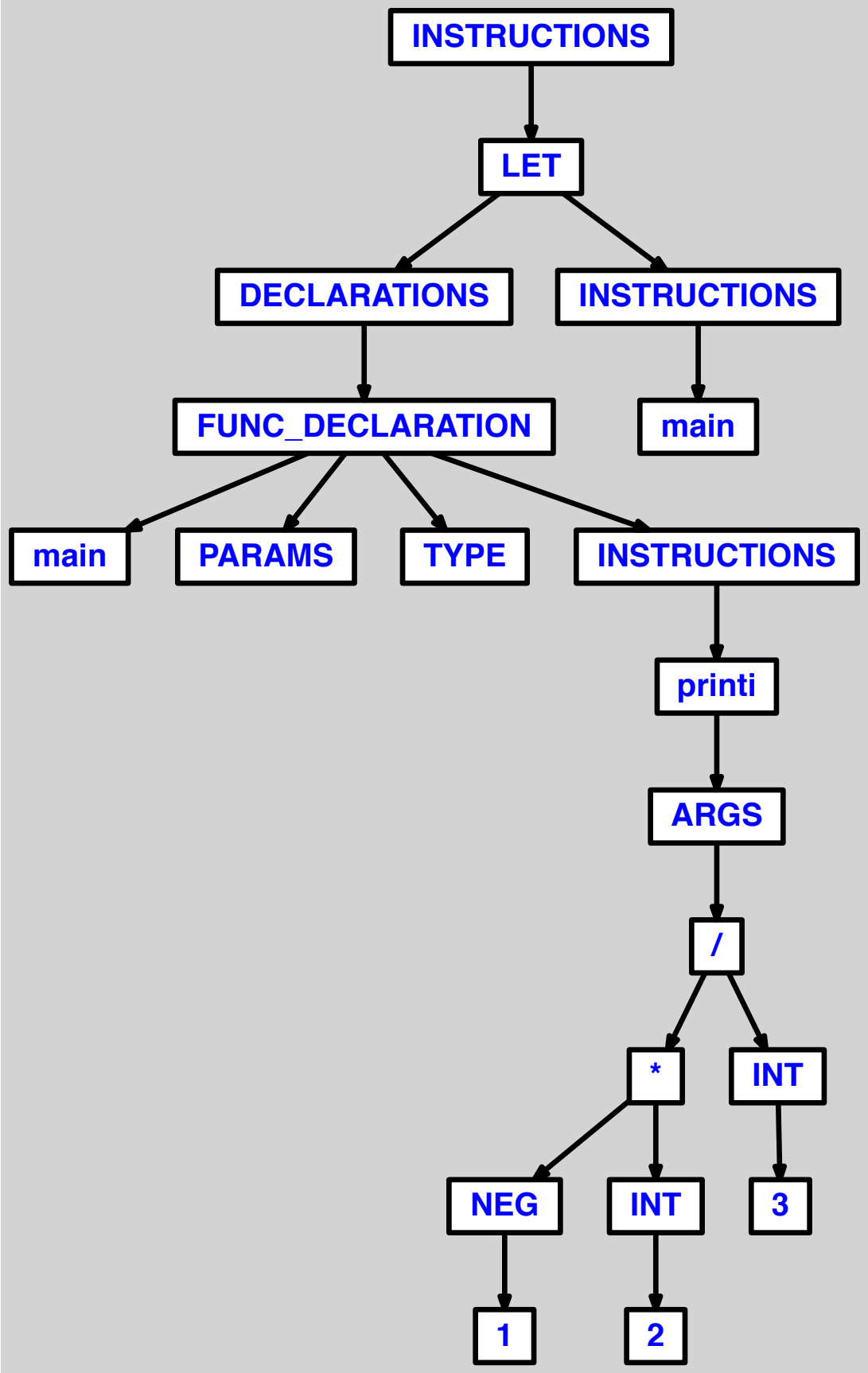
3.2.35 multiplication a 3 termes, dont un ayant moins unaire

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


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

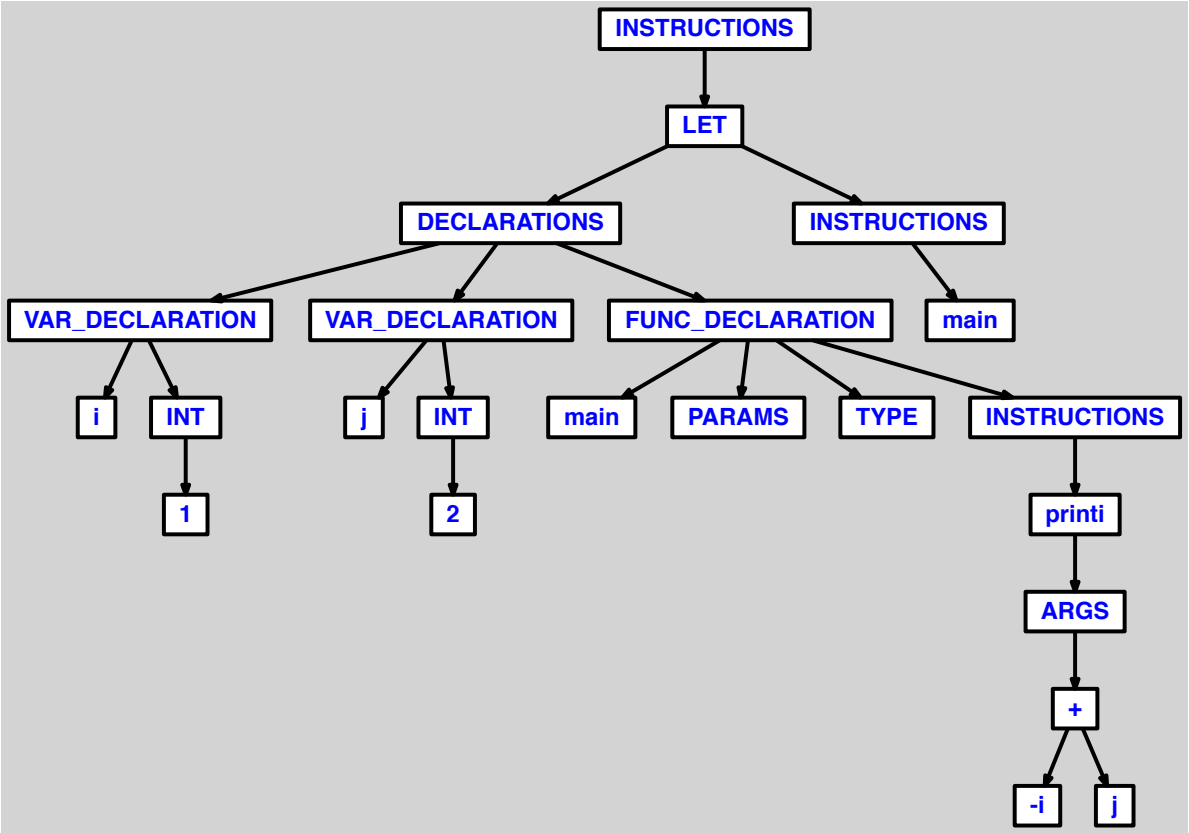
3.2.36 multiplication suivie de division, dont un terme ayant moins unaire

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


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

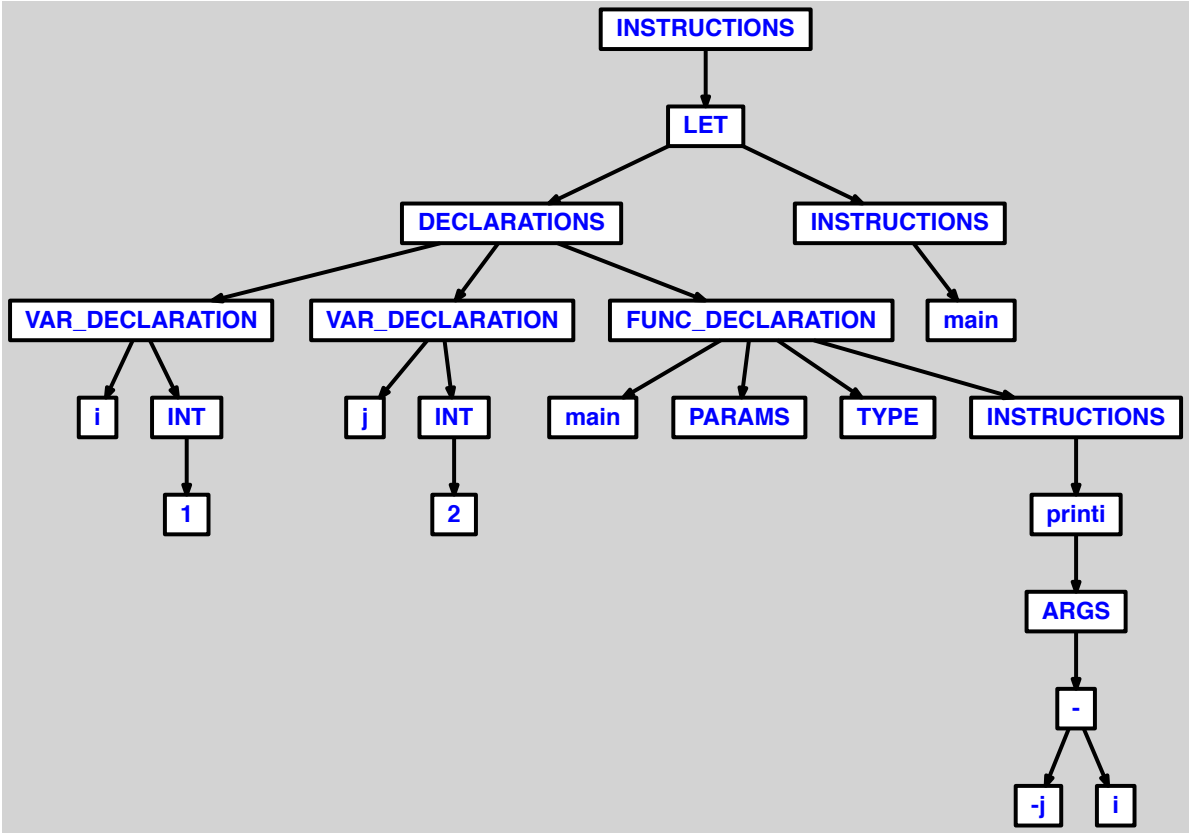
3.2.37 addition simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

```
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
main
i
INT
j
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
printi
ARGS
+
-i
j

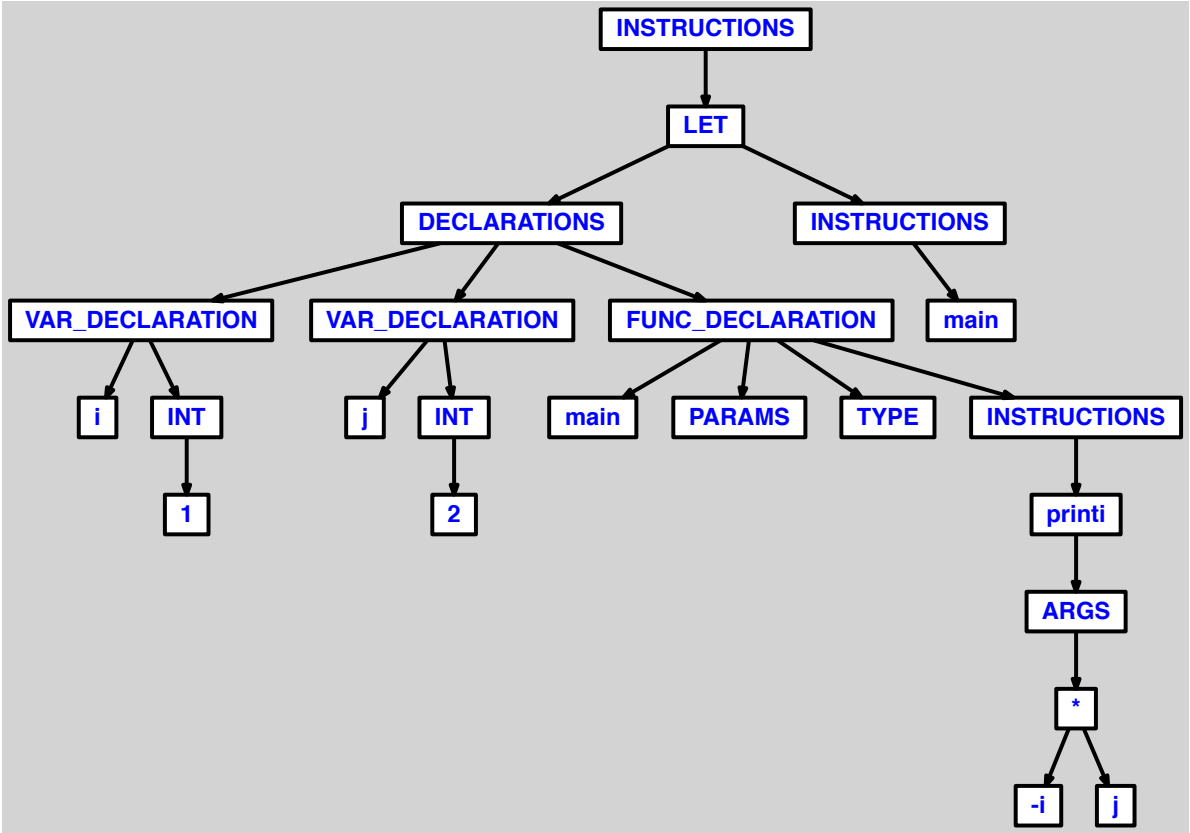
3.2.38 soustraction simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

```
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
main
i
INT
j
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
printi
ARGS
-
-j
i

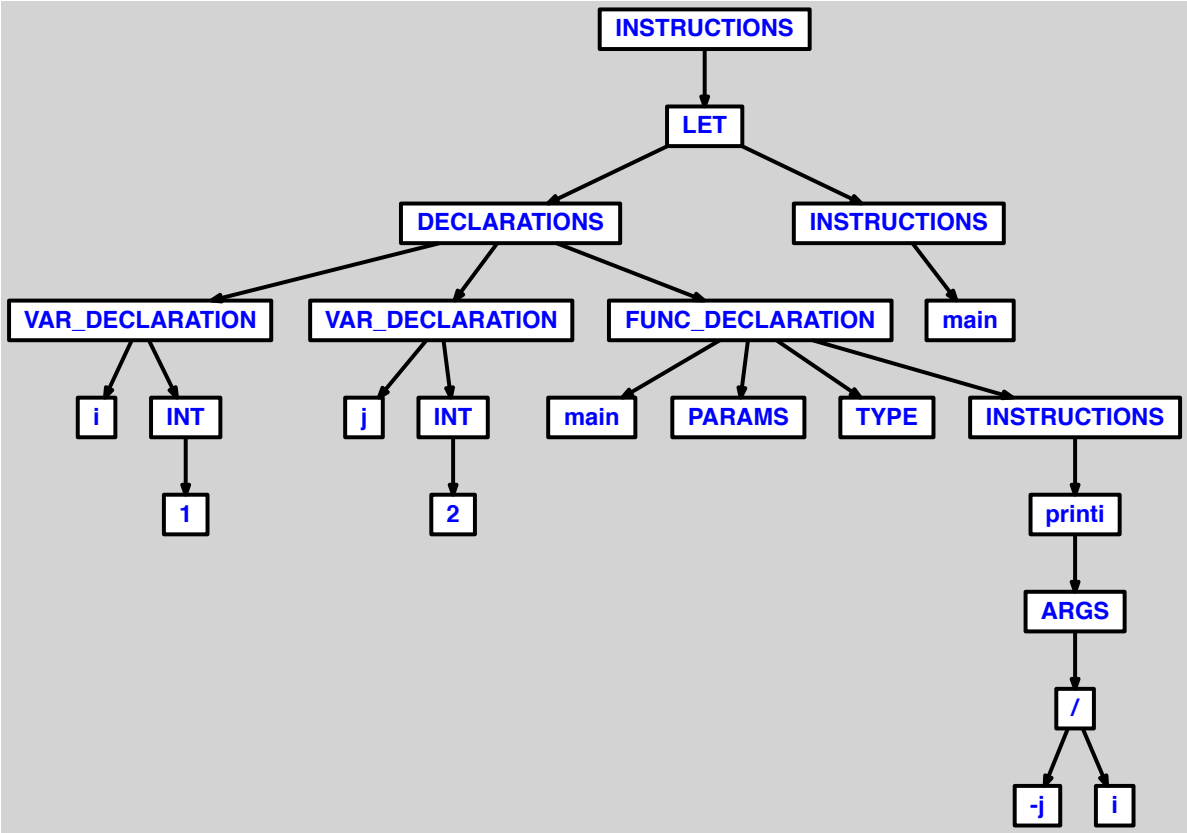
3.2.39 multiplication simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

```
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
main
i
INT
j
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
printi
ARGS
*
-i
j

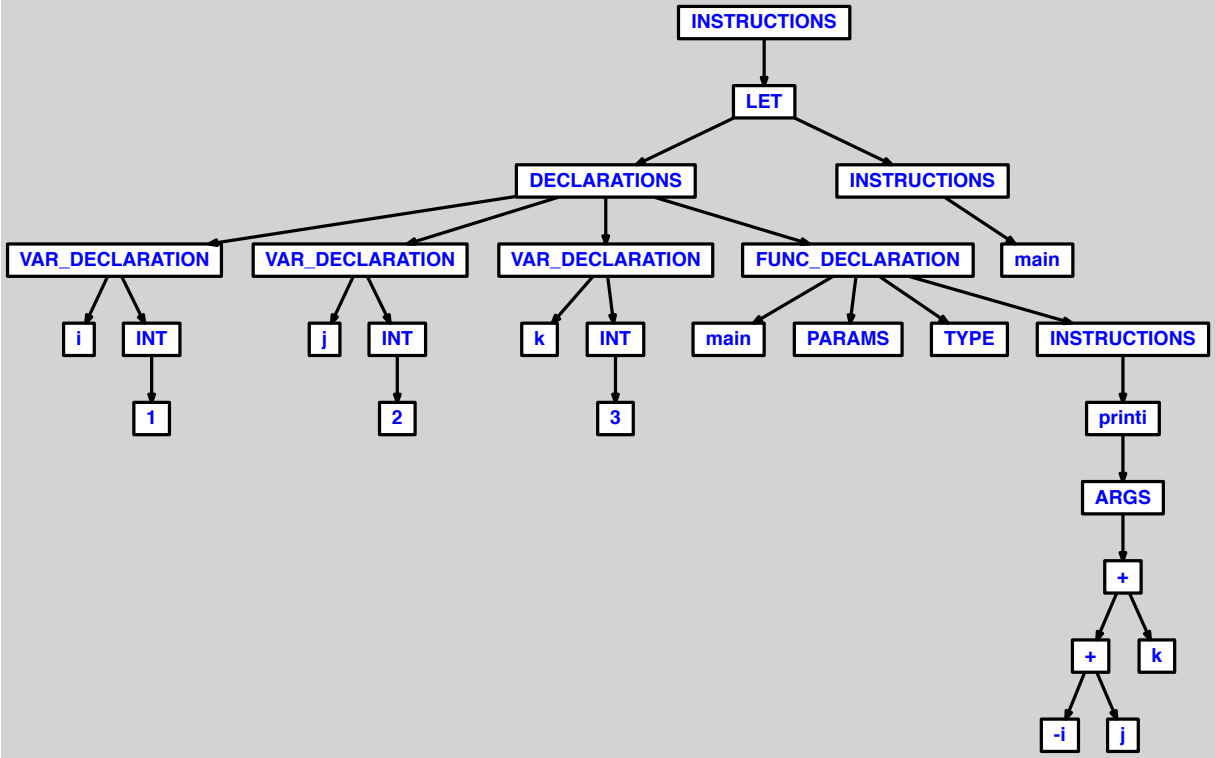
3.2.40 division simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

```
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
main
i
INT
j
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
printi
ARGS
/
-j
i

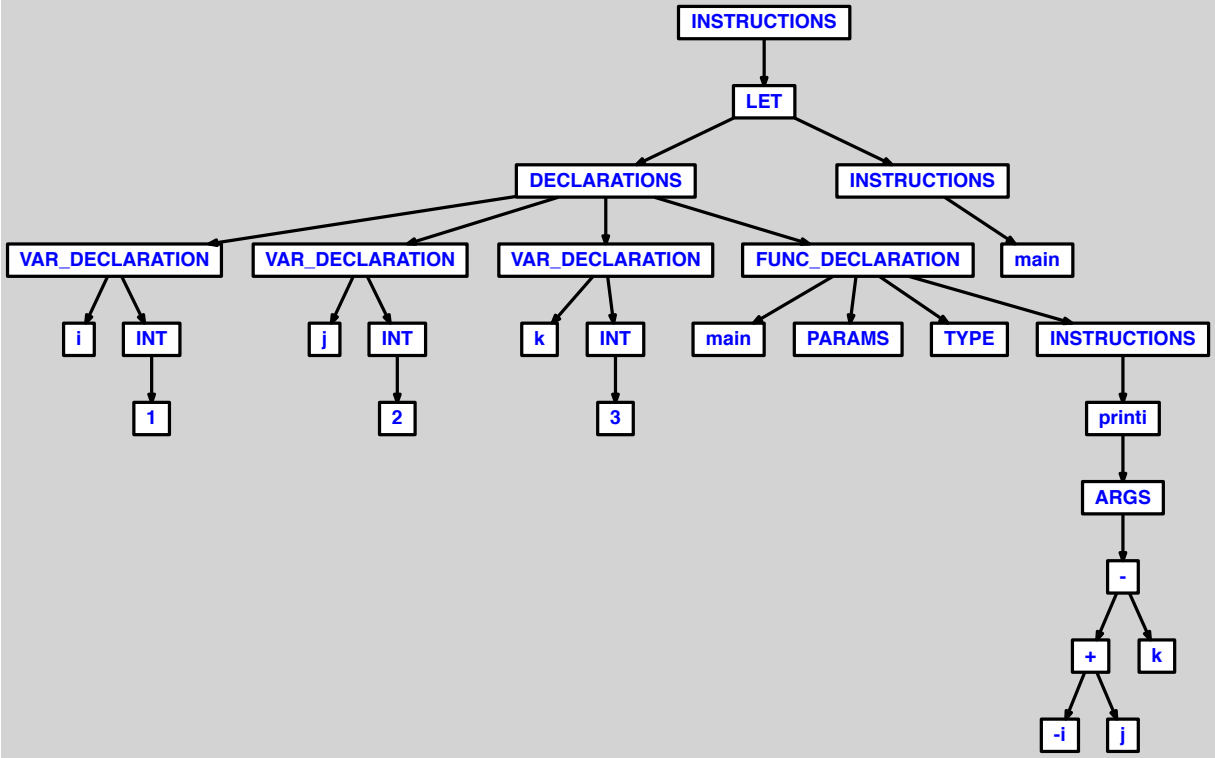
3.2.41 addition a 3 termes, identifies par des variables, dont une ayant moins unaire

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
+
k
-i
j

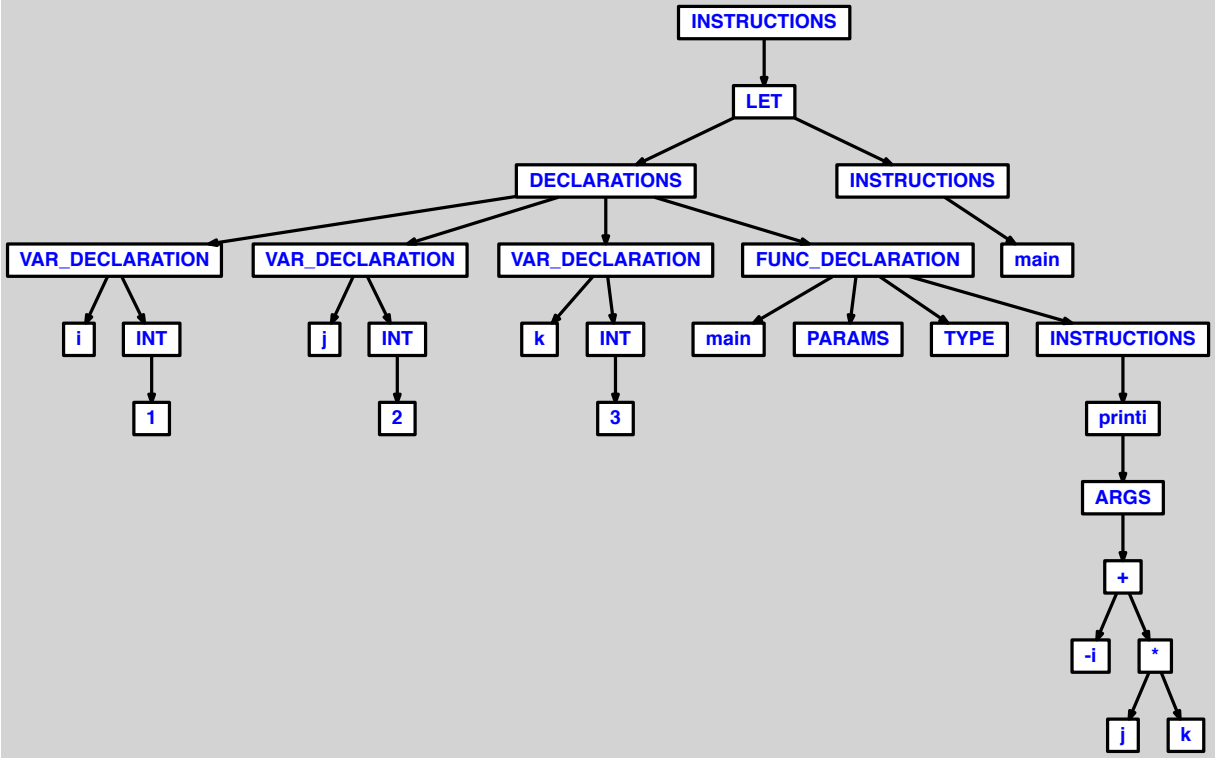
3.2.42 addition suivie de soustraction, avec termes identifies par variables, dont une ayant moins unaire

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
-
+
k
-i
j

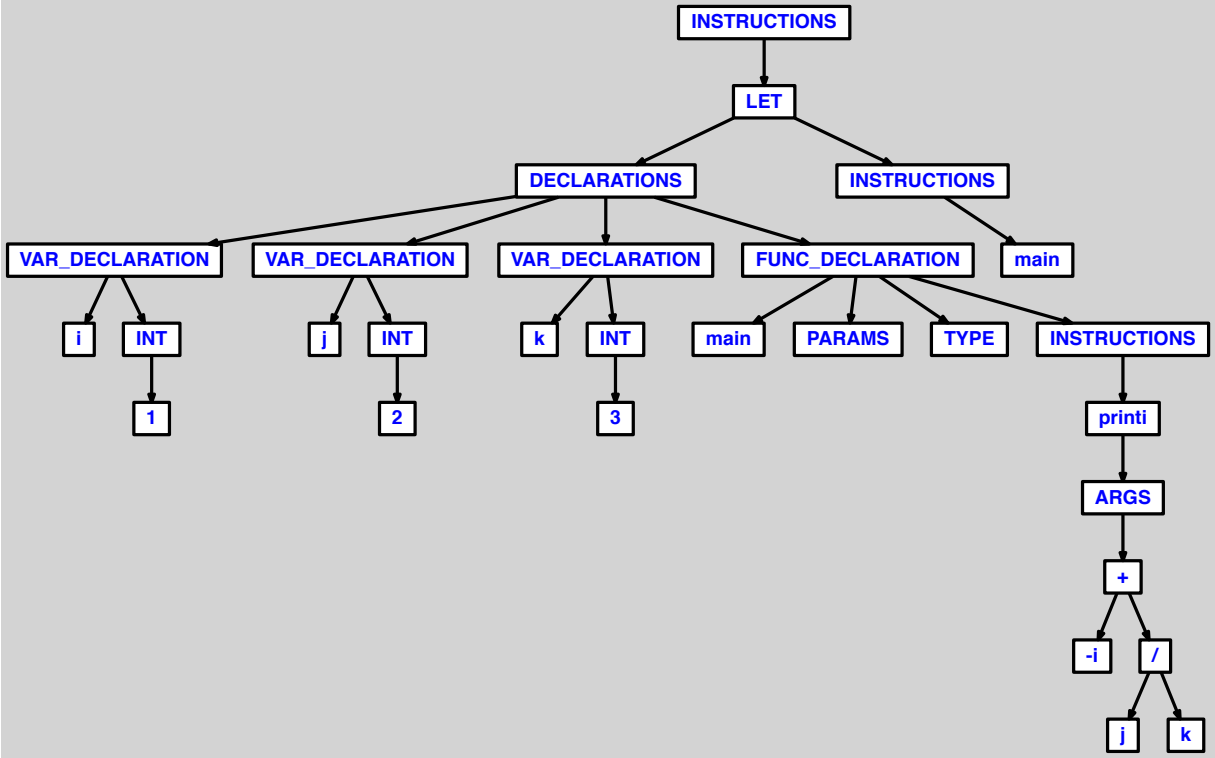
3.2.43 addition suivie de multiplication, avec termes identifies par variables, dont une ayant moins unaire

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
-i
*
j
k

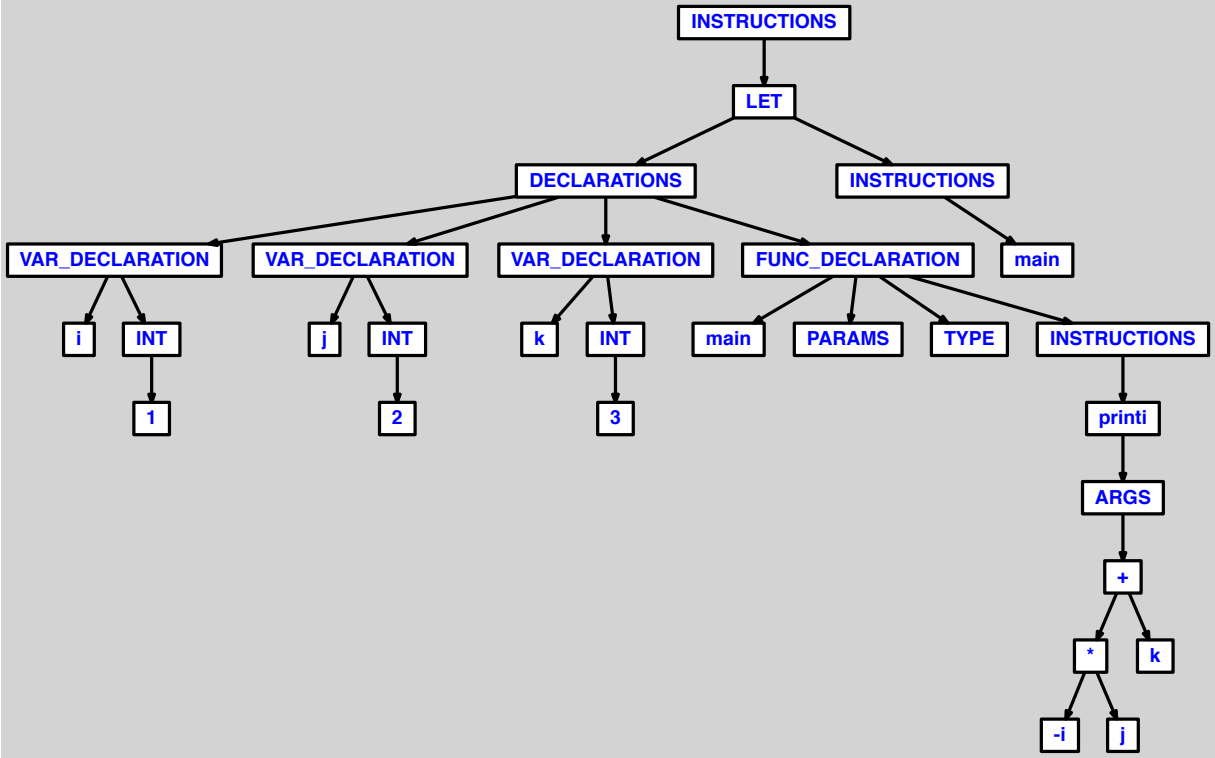
3.2.44 addition suivie de division, avec termes identifies par variables, dont une ayant moins unaire

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
-i
/
j
k

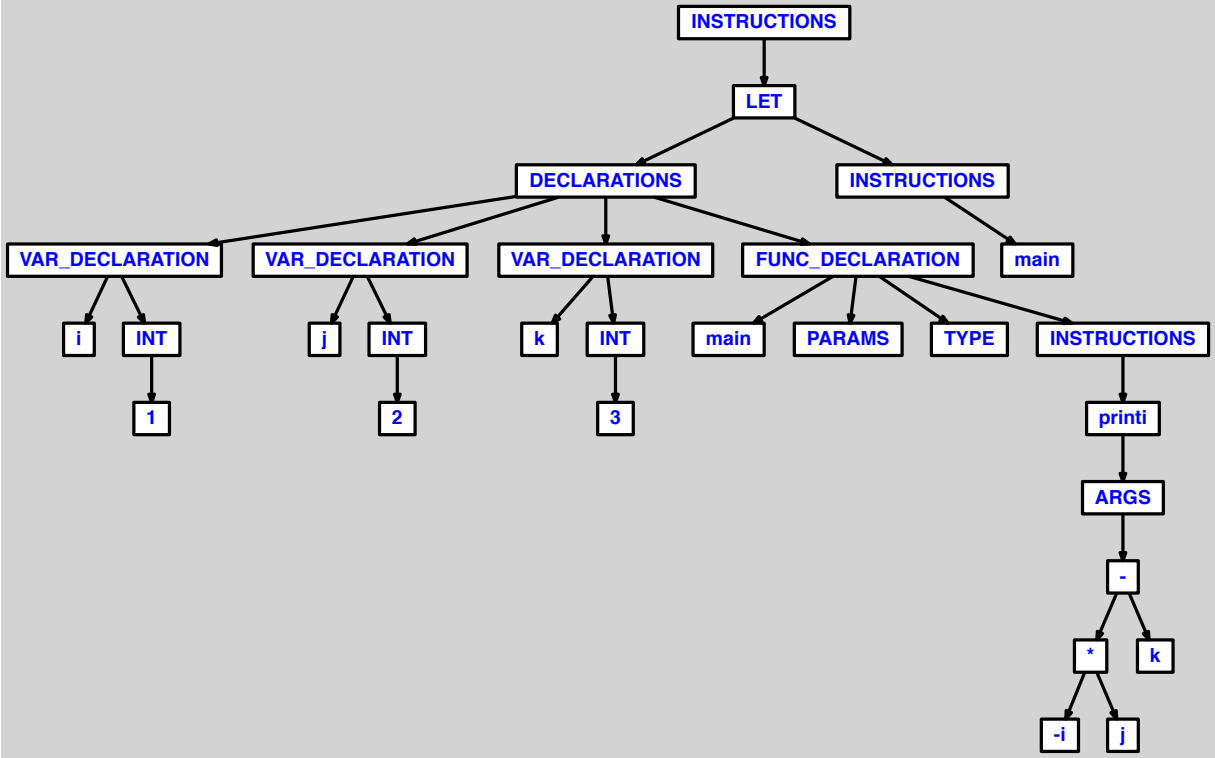
3.2.45 multiplication suivie d'addition, avec termes identifies par variables, dont une ayant moins unaire

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
*
k
-i
j

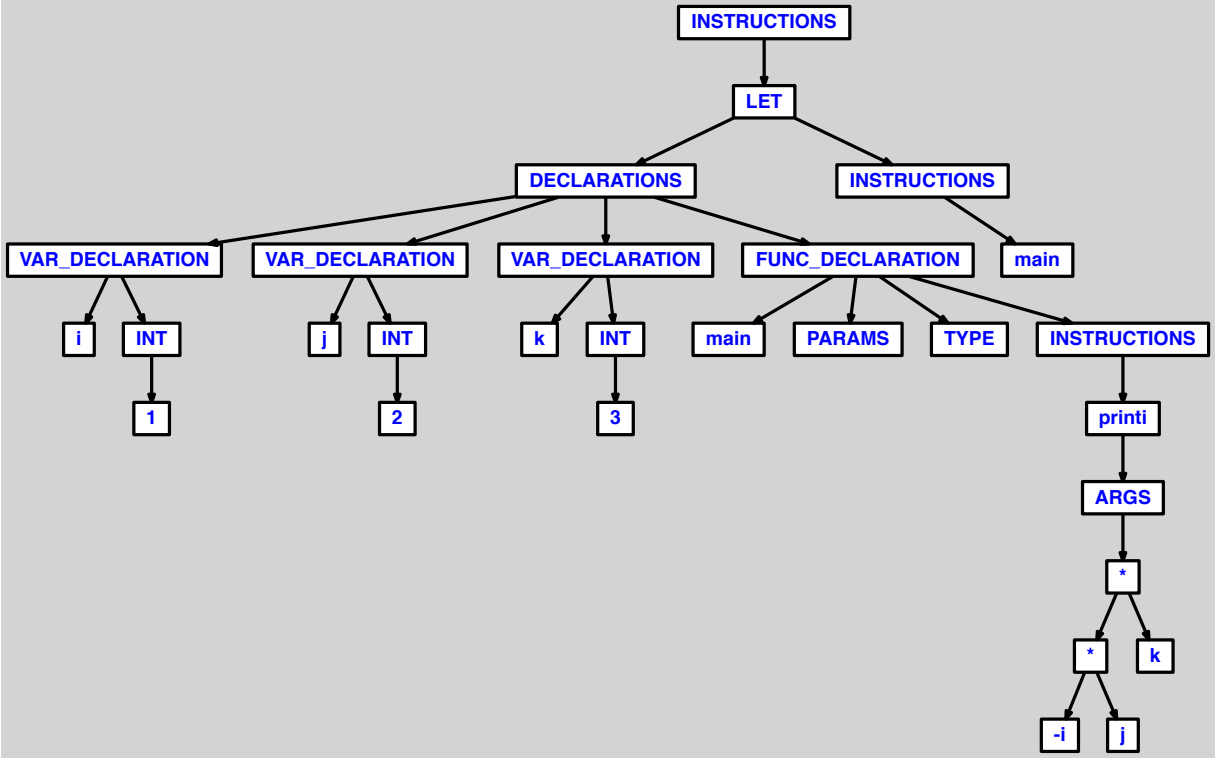
3.2.46 multiplication suivie de soustraction, avec termes identifies par variables, dont une ayant moins unaire

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
-
*
k
-i
j

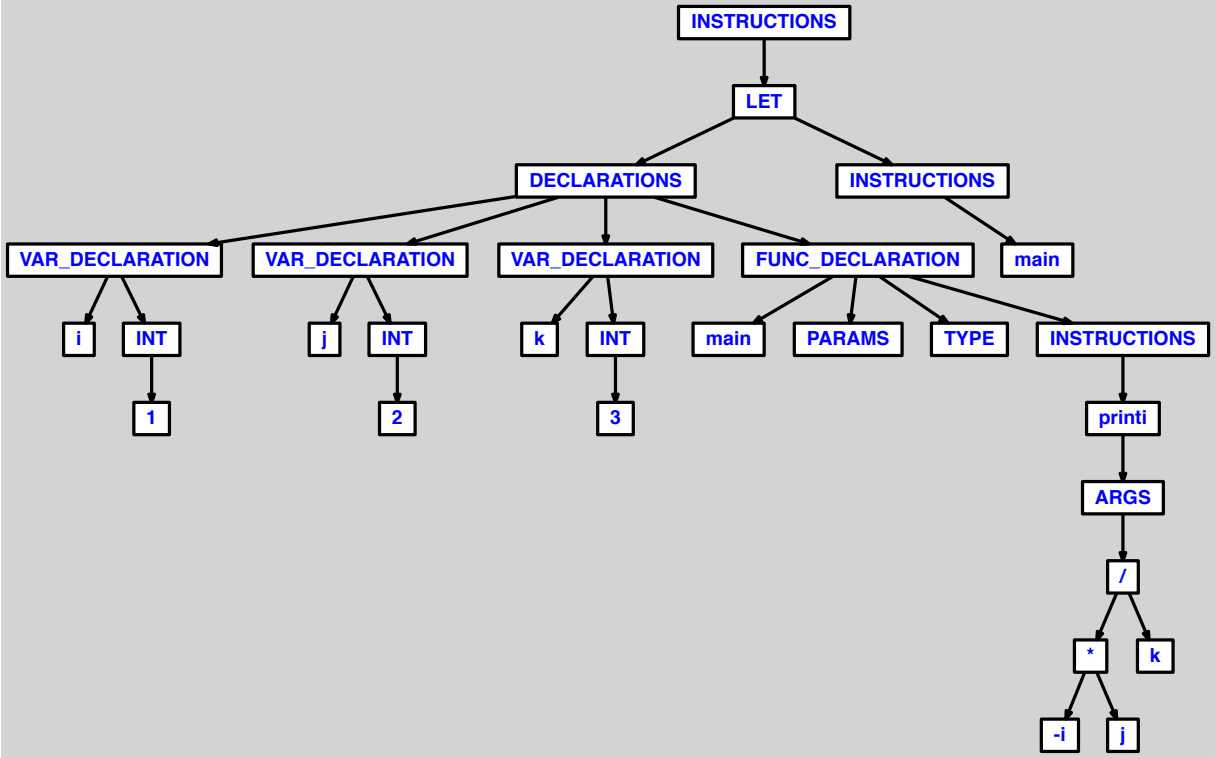
3.2.47 multiplication a 3 termes, identifies par des variables, dont une ayant moins unaire

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
*
*
k
-i
j

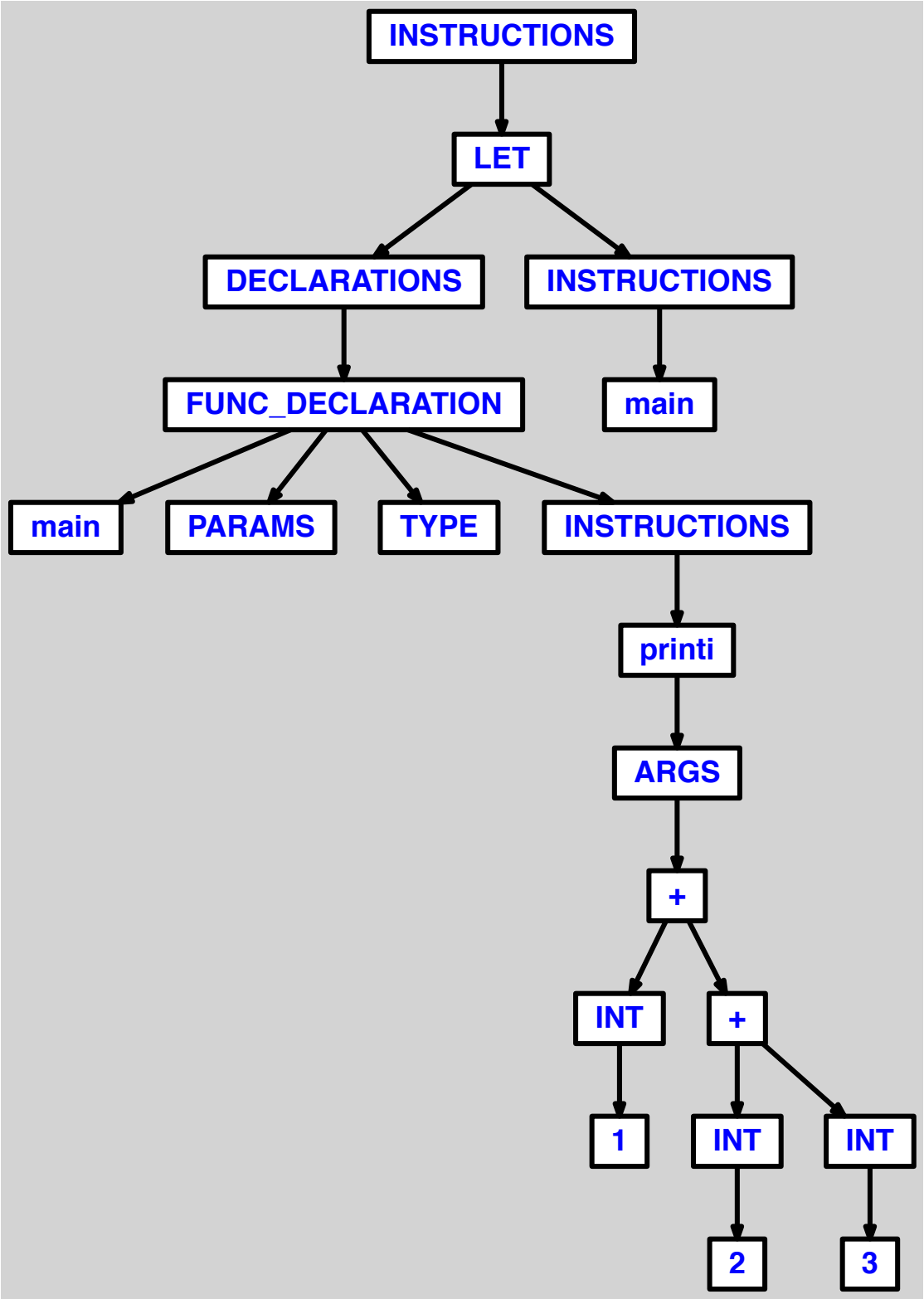
3.2.48 mutliplication suivie de division, avec termes identifies par variables, dont une ayant moins unaire

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
/
*
k
-i
j

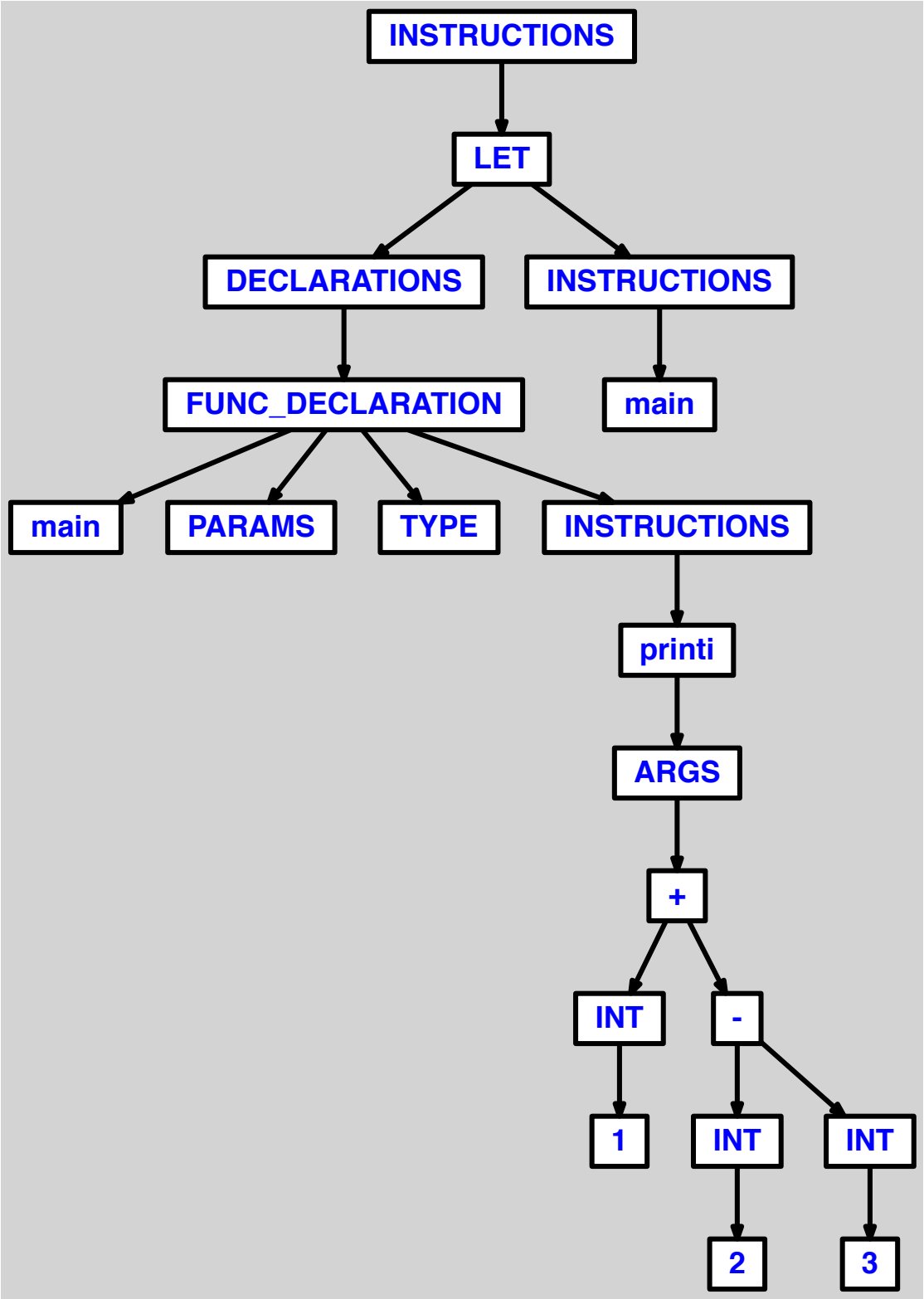
3.2.49 addition a 3 termes, avec parenthesage des 2 termes a droite

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


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

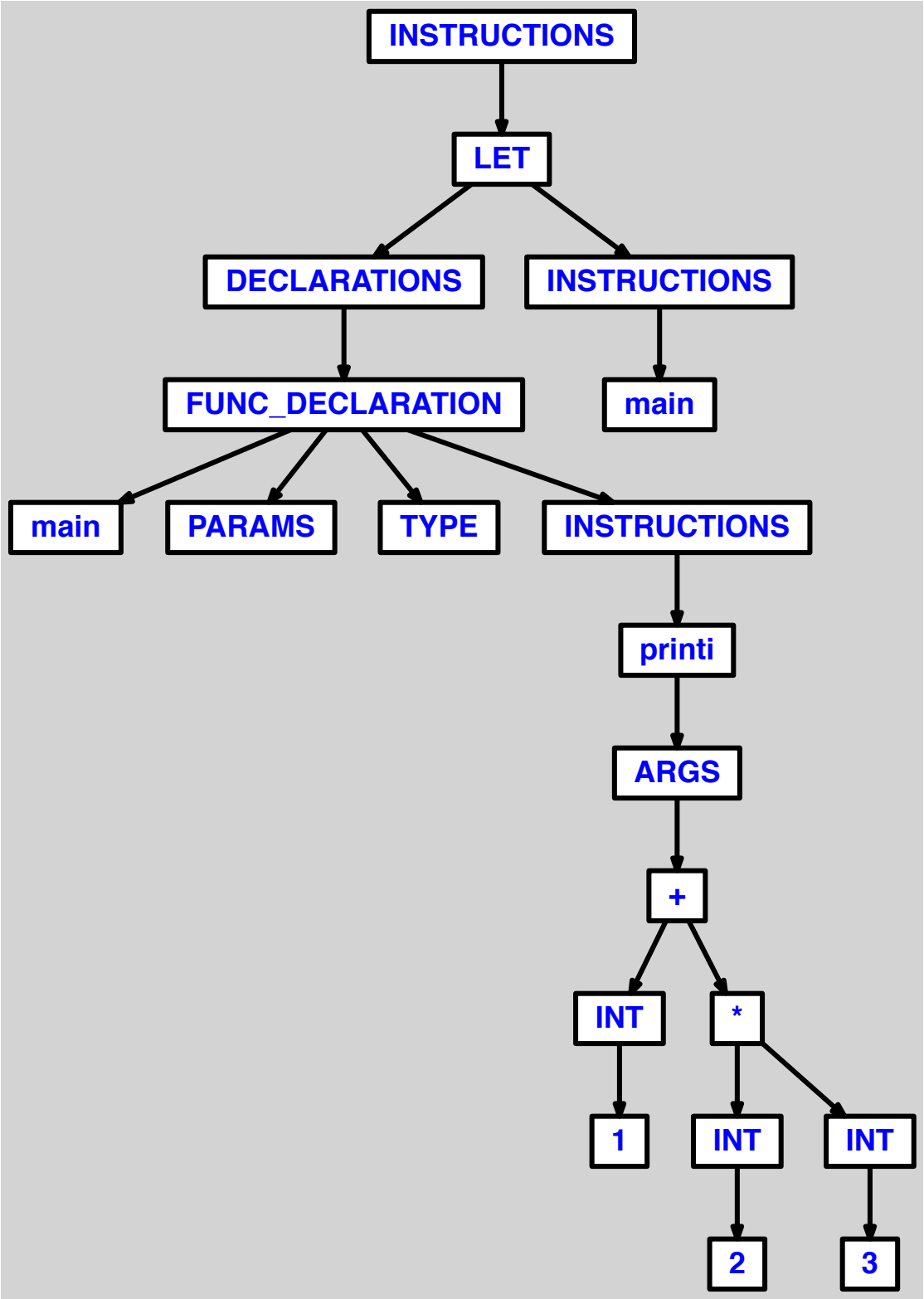
3.2.50 addition suivie de soustraction, avec parenthesage des 2 termes a droite

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


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

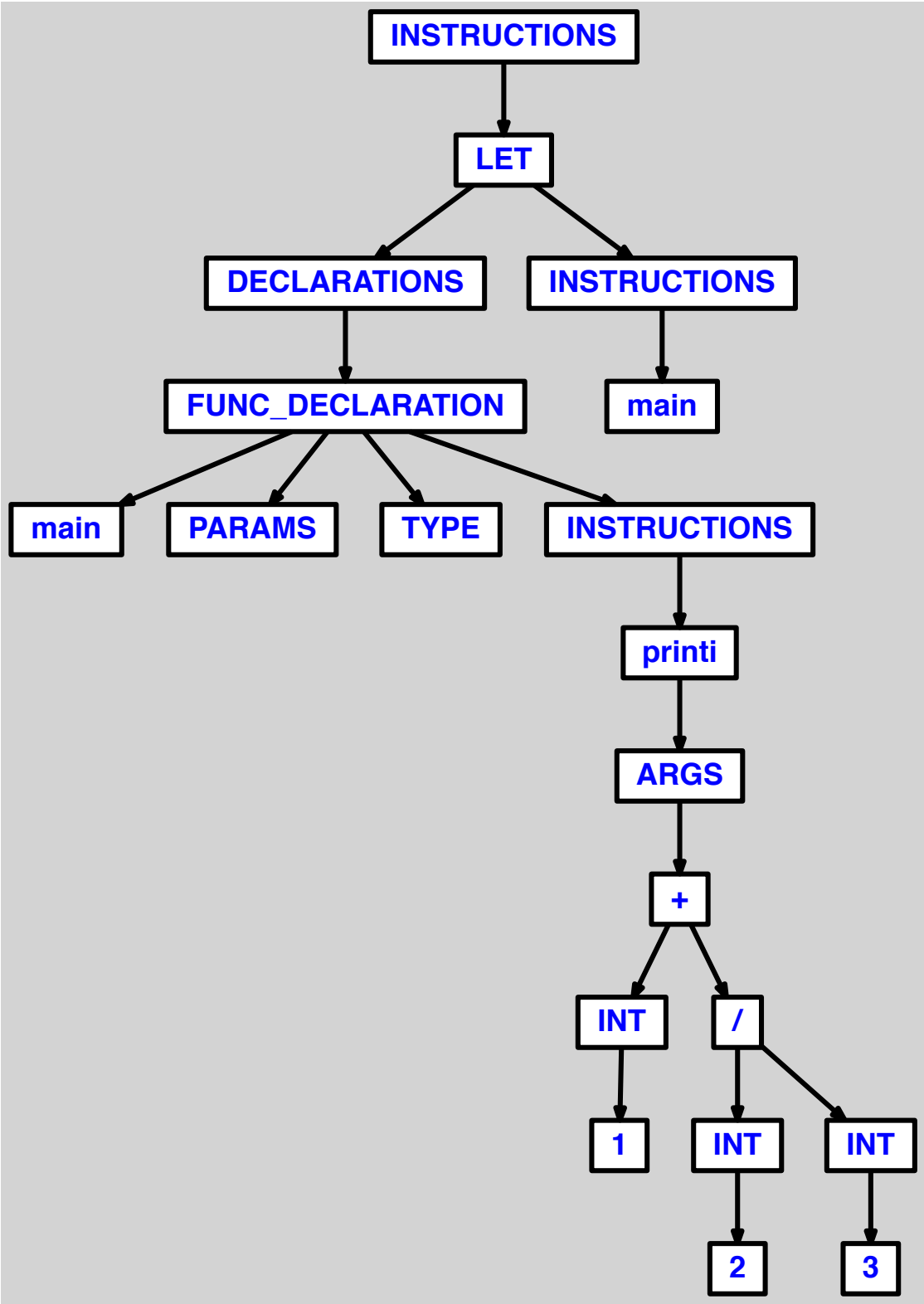
3.2.51 addition suivie de multiplication, avec parenthesage des 2 termes a droite

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


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

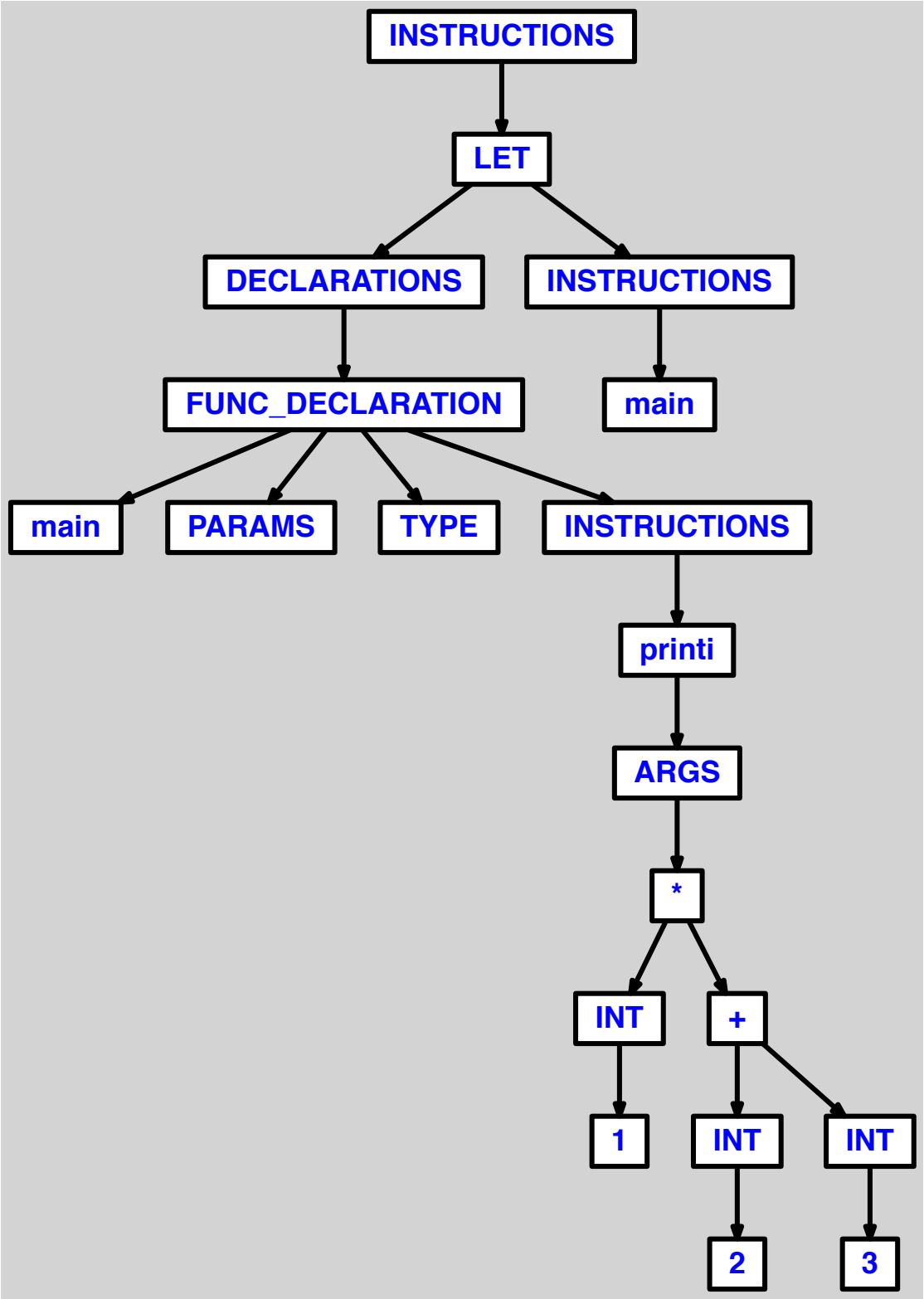
3.2.52 addition suivie de division, avec parenthesage des 2 termes a droite

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


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

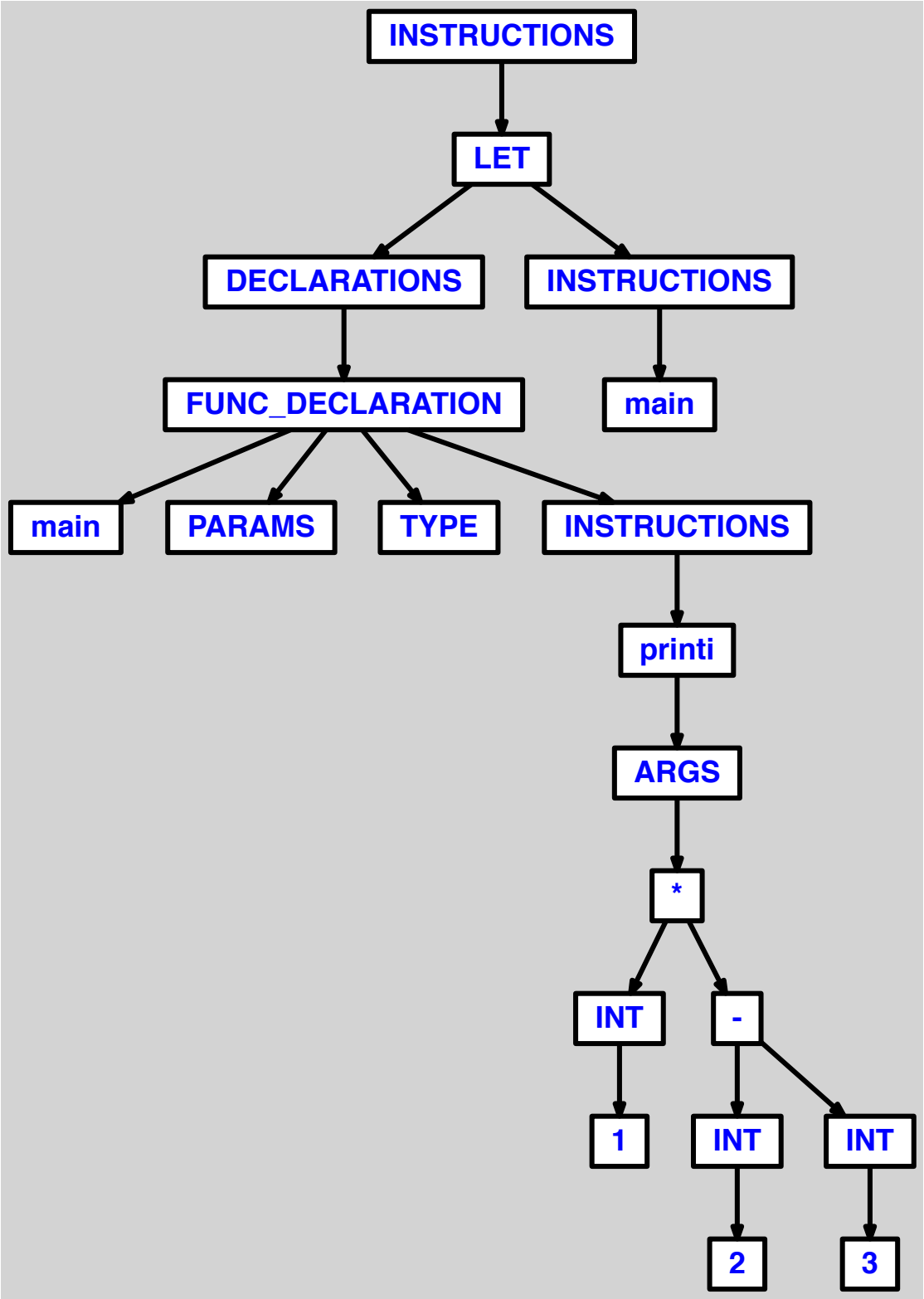
3.2.53 multiplication suivie d'addition, avec parenthesage des 2 termes a droite

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


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

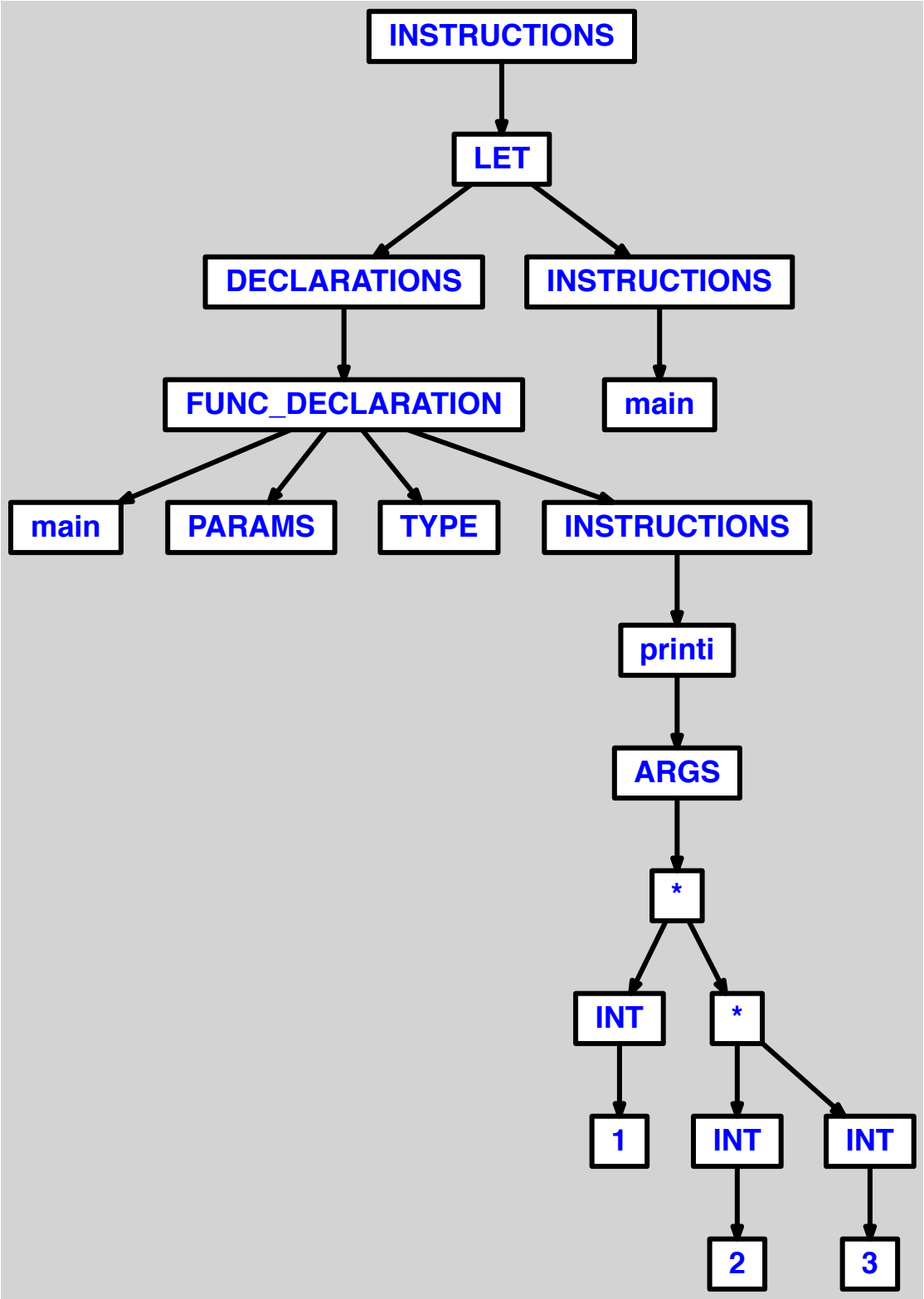
3.2.54 multiplication suivie de soustraction, avec parenthesage des 2 termes a droite

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


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

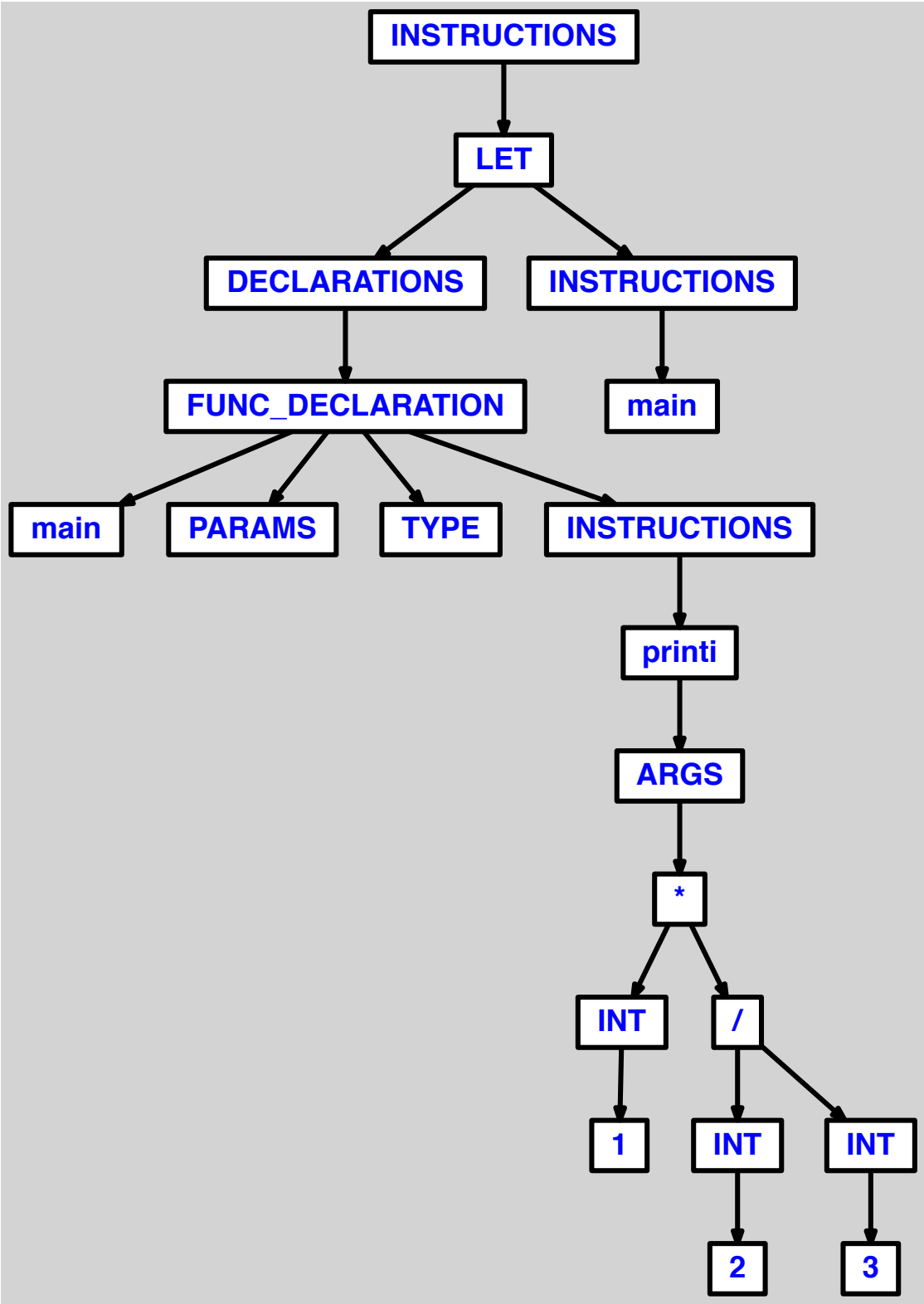
3.2.55 multiplication a 3 termes, avec parenthesage des 2 termes a droite

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


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

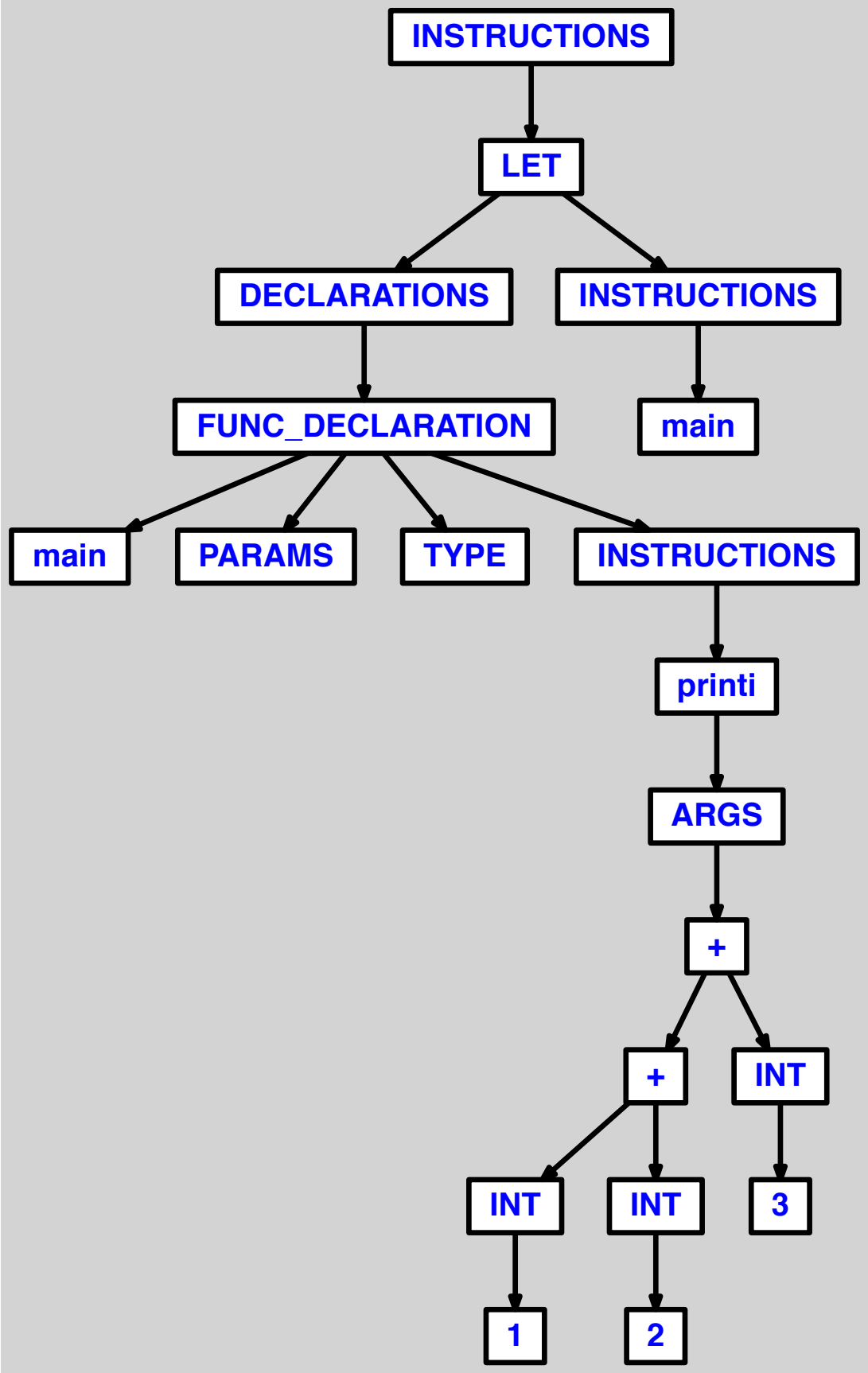
3.2.56 multiplication suivie de vision, avec parenthesage des 2 termes a droite

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


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

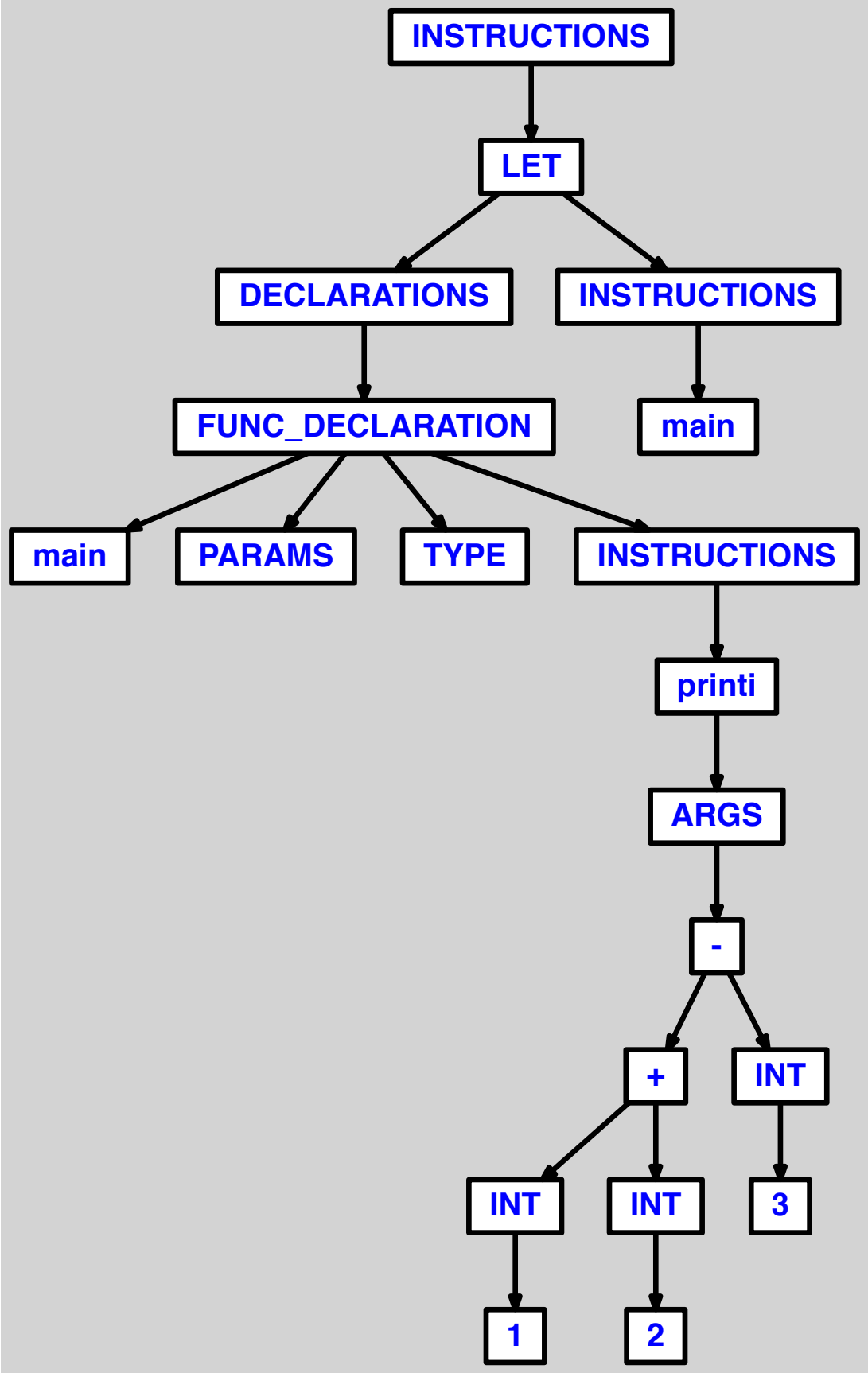
3.2.57 addition a 3 termes, avec parenthesage des 2 termes a gauche

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


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

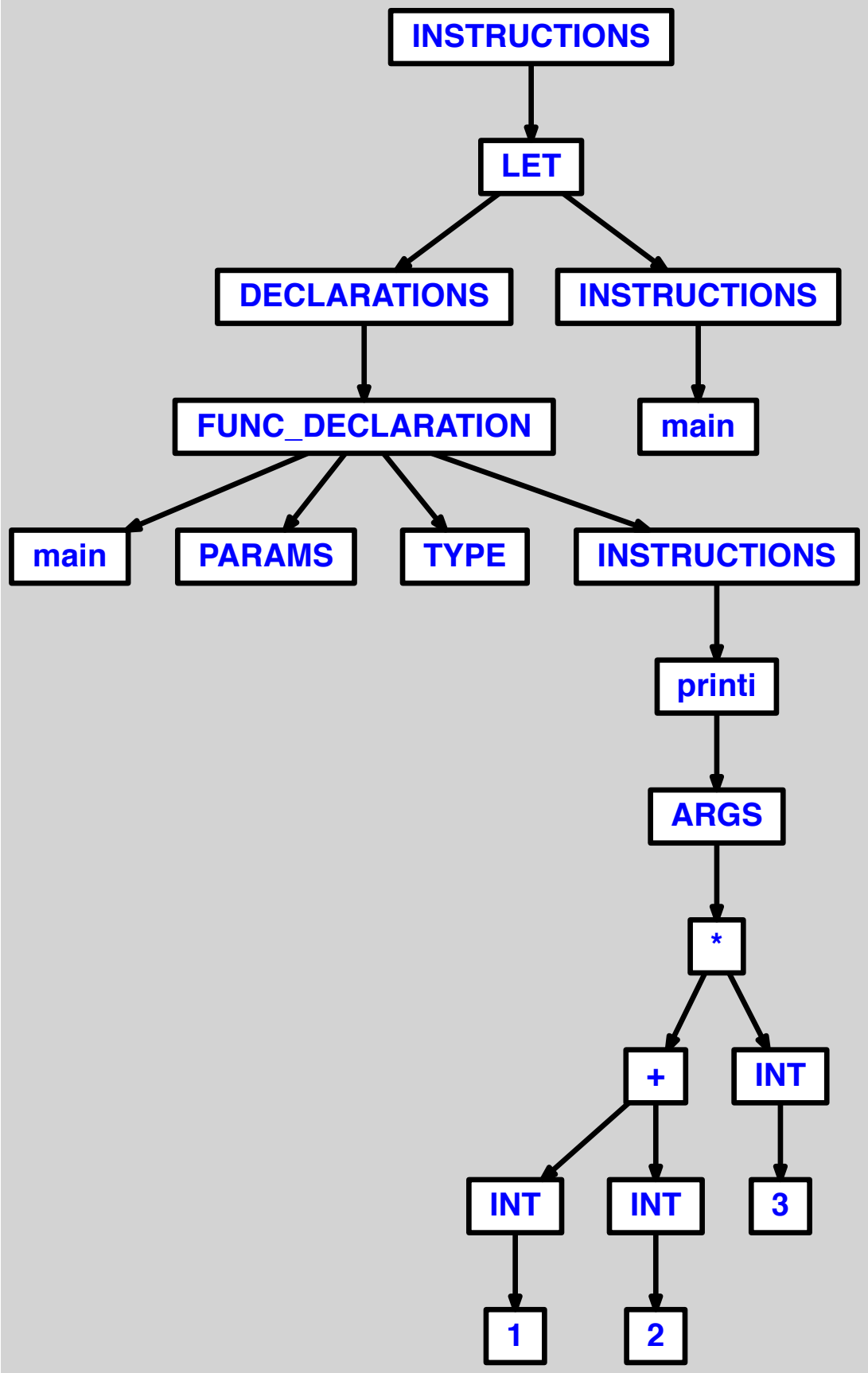
3.2.58 addition suivie de soustraction, avec parenthesage des 2 termes a gauche

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


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

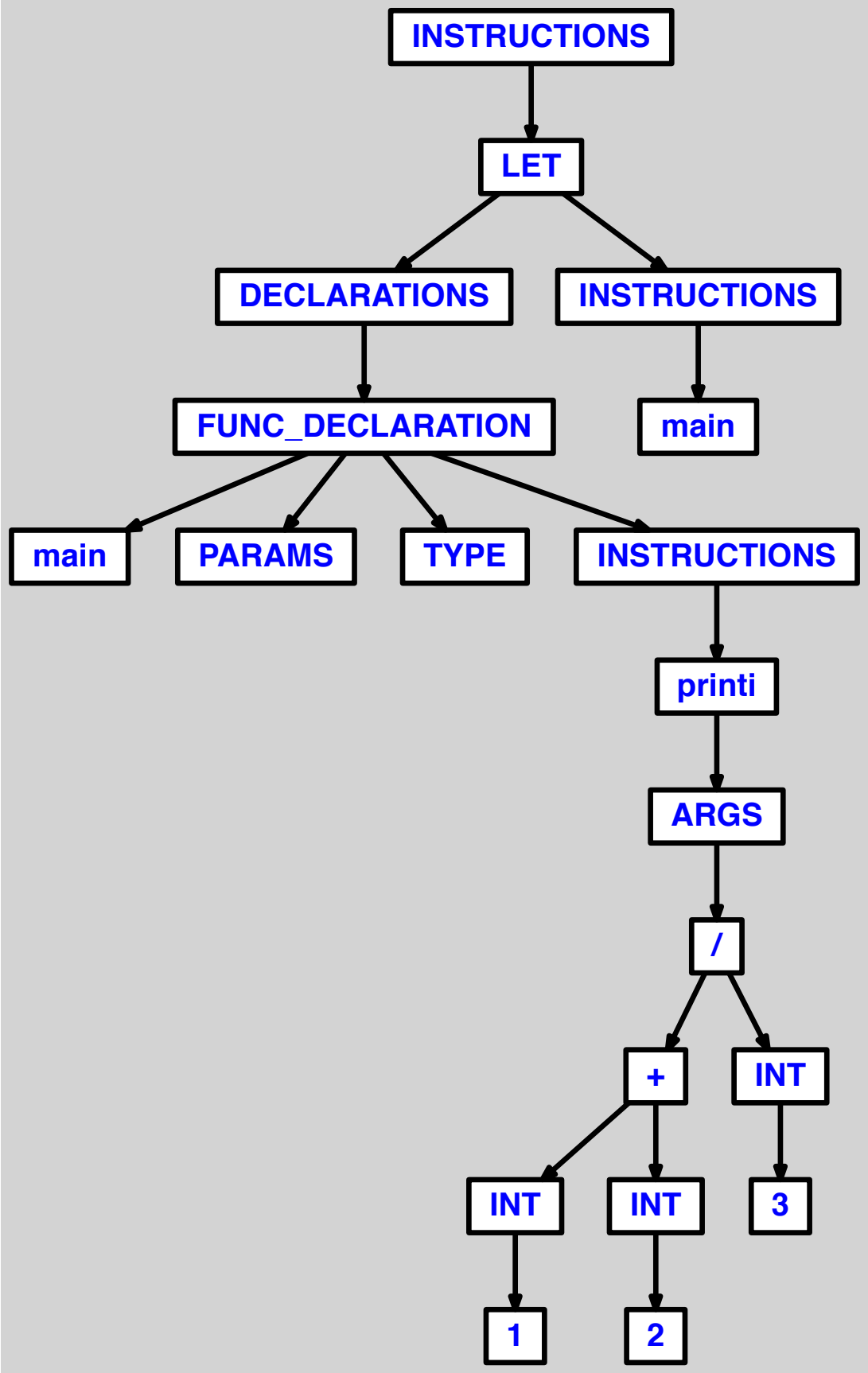
3.2.59 addition suivie de multiplication, avec parenthesage des 2 termes a gauche

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


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

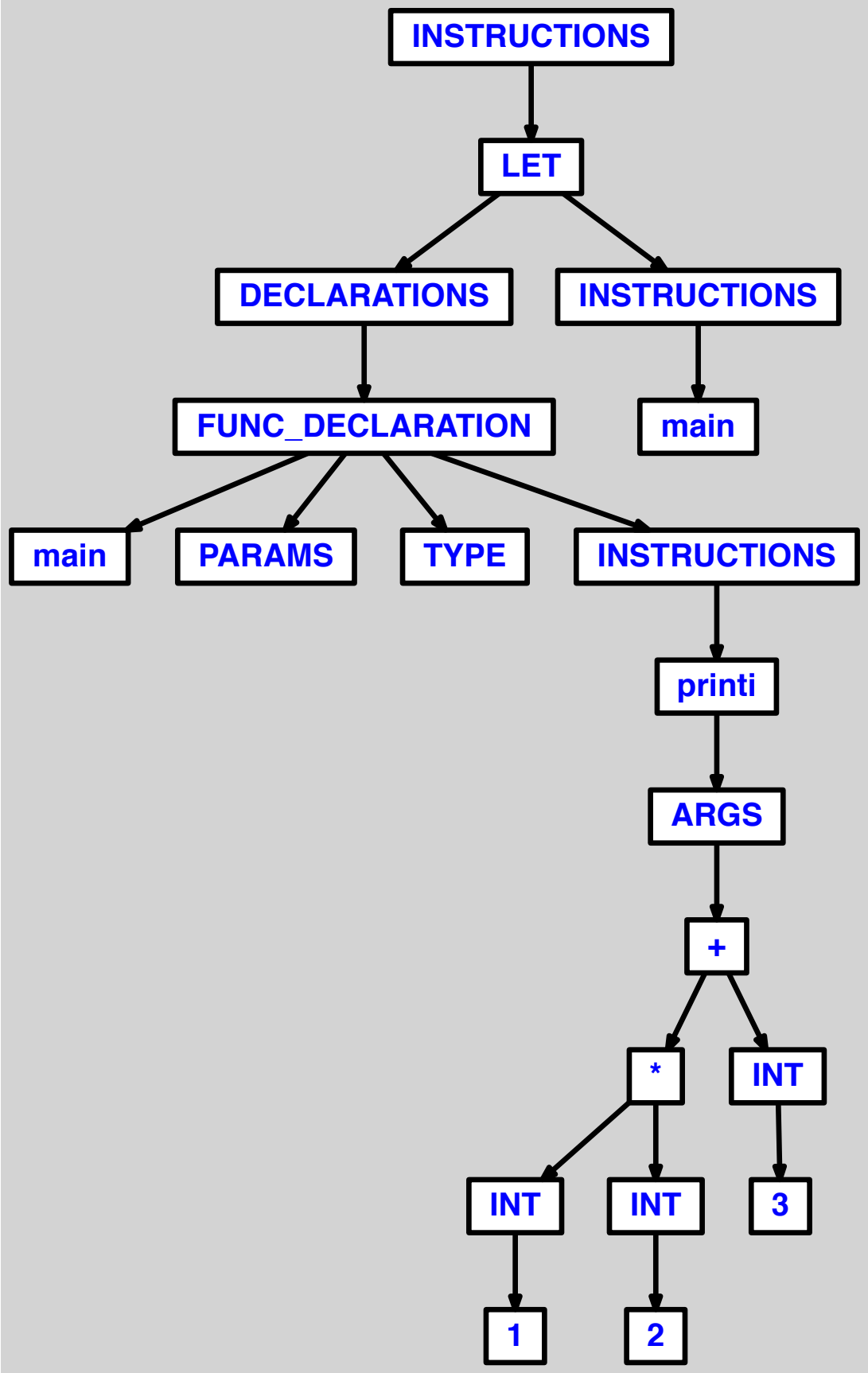
3.2.60 addition suivie de division, avec parenthesage des 2 termes a gauche

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


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

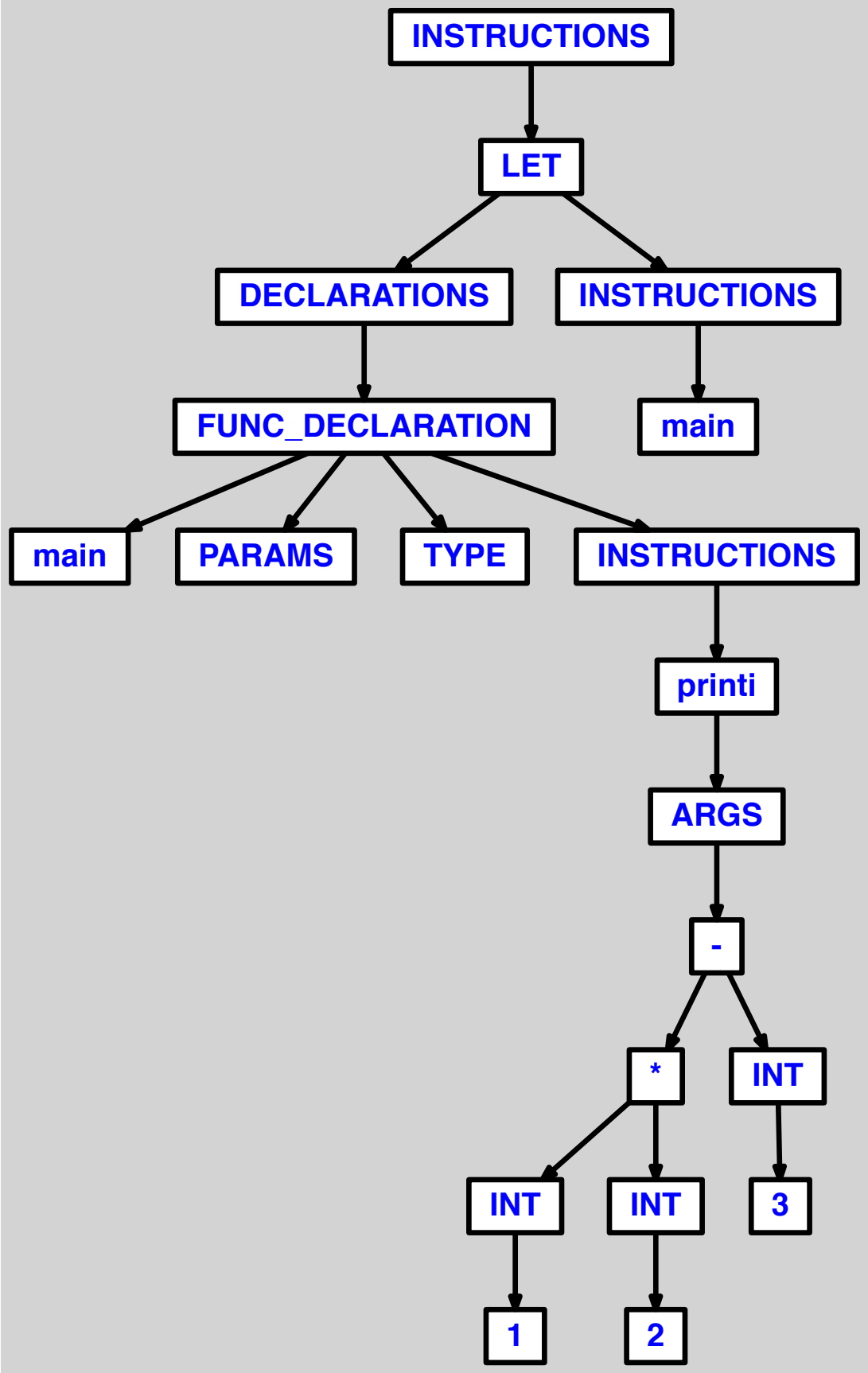
3.2.61 multiplication suivie d'addition, avec parenthesage des 2 termes a gauche

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


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

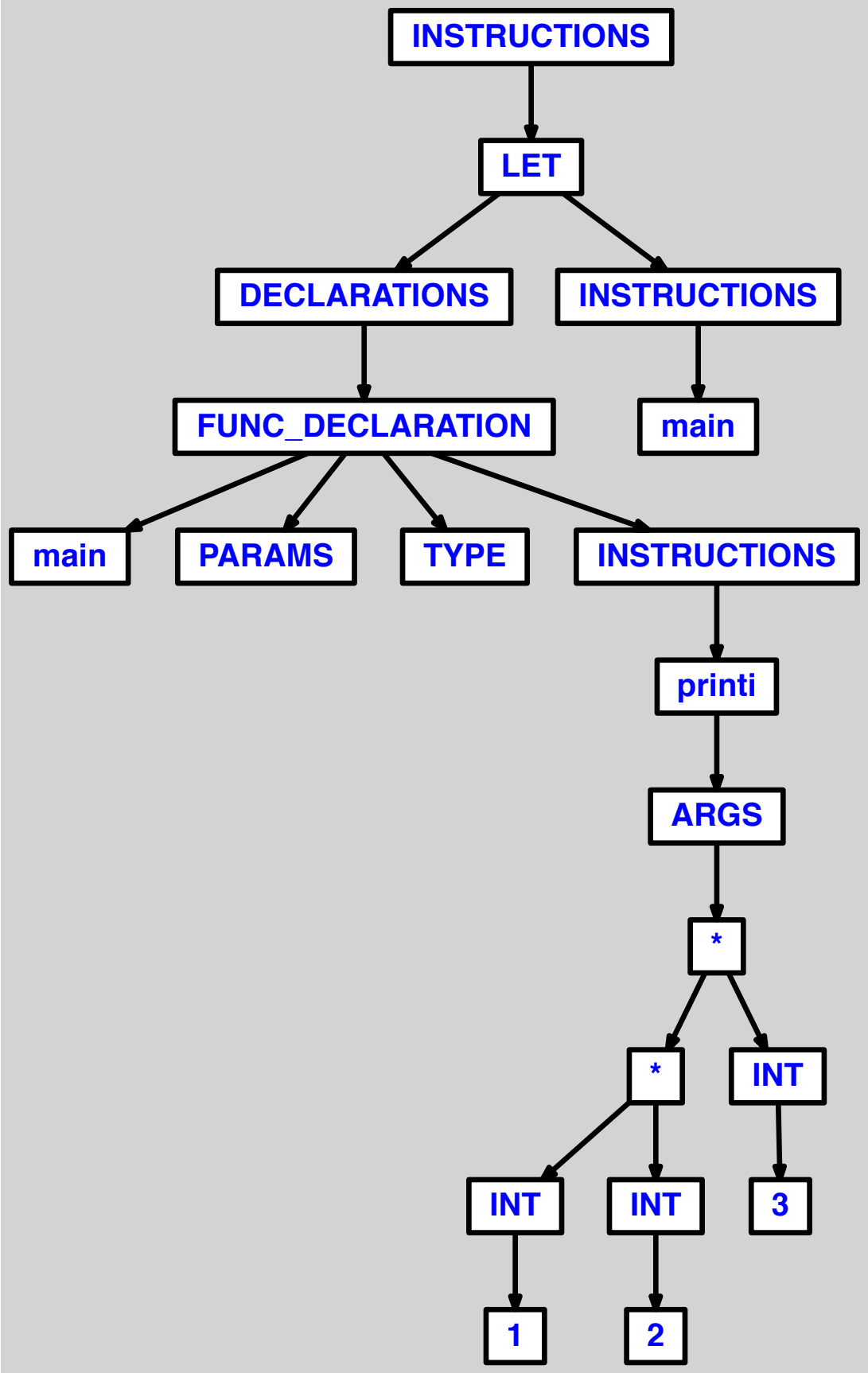
3.2.62 multiplication suivie de soustraction, avec parenthesage des 2 termes a gauche

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


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

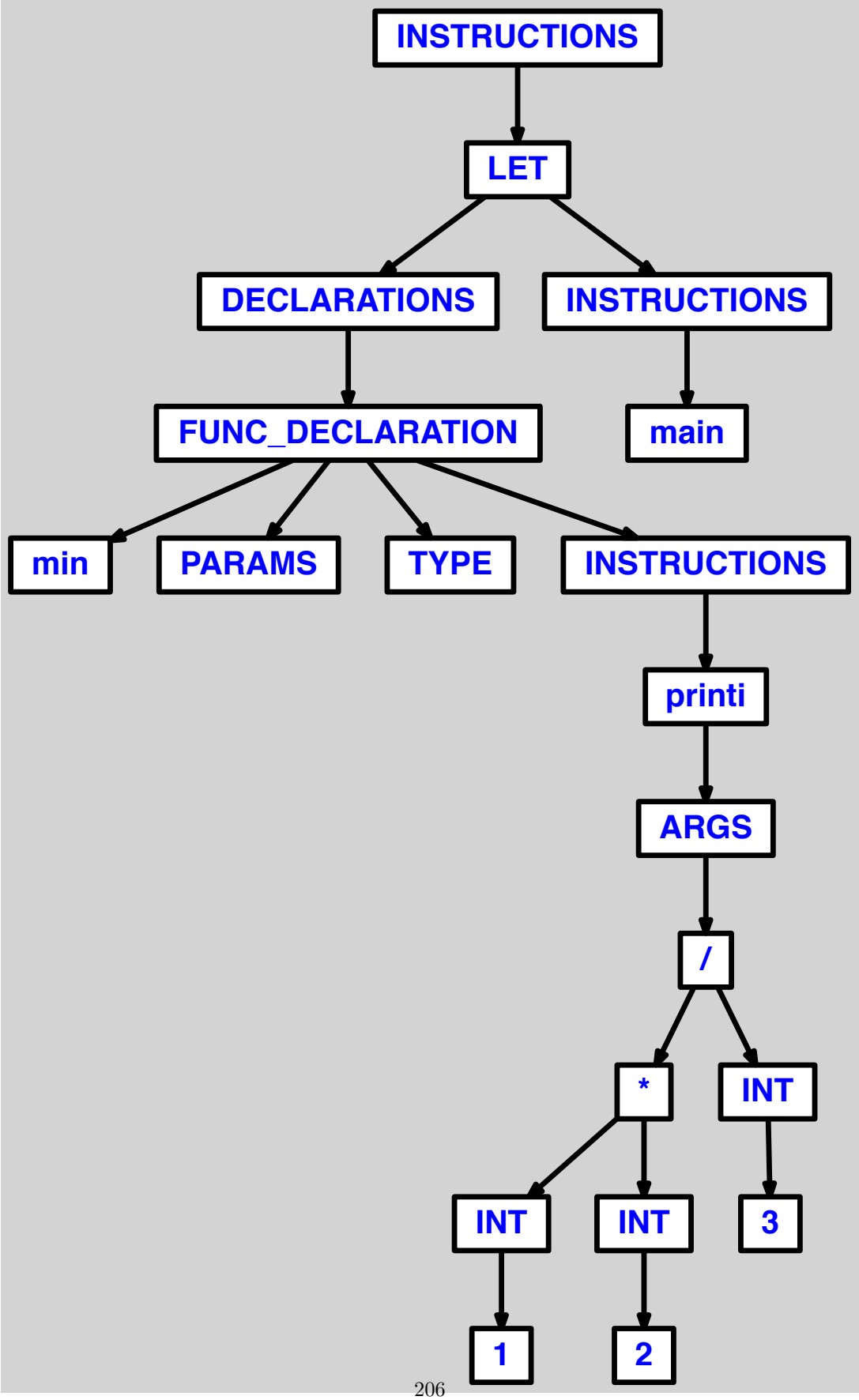
3.2.63 multiplication a 3 termes, avec parenthesage des 2 termes a gauche

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


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

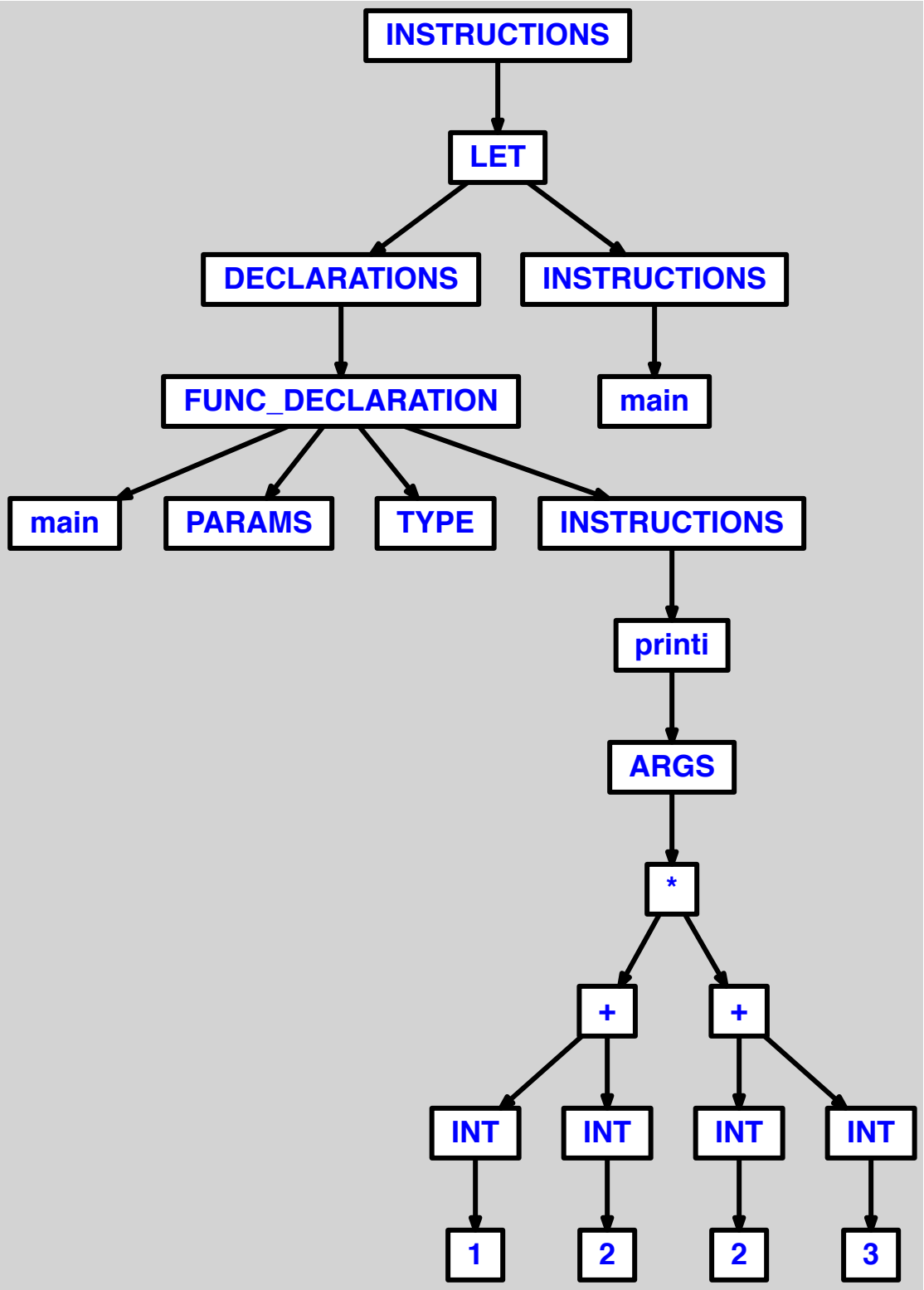
3.2.64 multiplication suivie de division, avec parenthesage des 2 termes a gauche

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


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

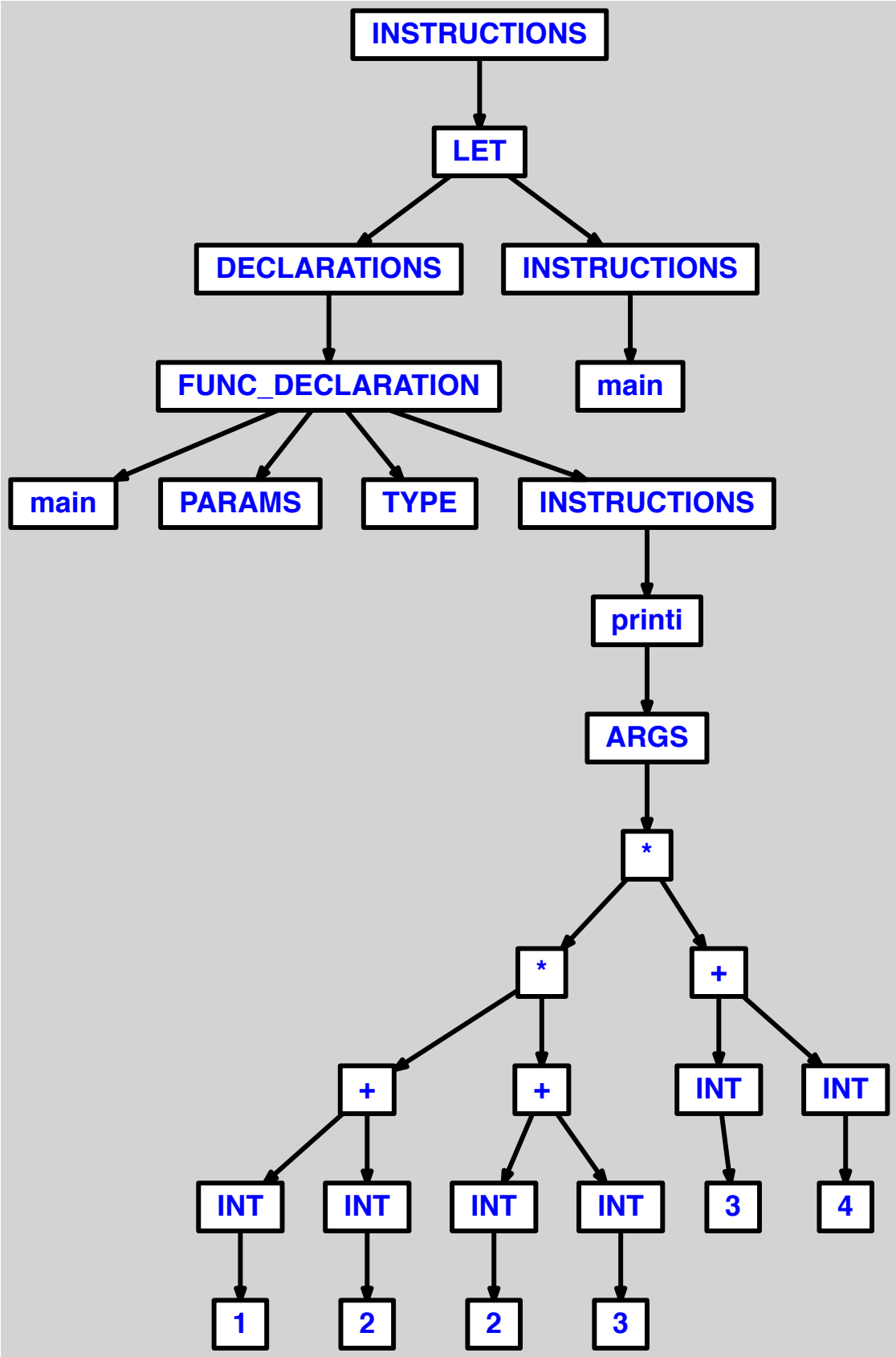
3.2.65 multiplication de 2 additions parenthesees

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


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

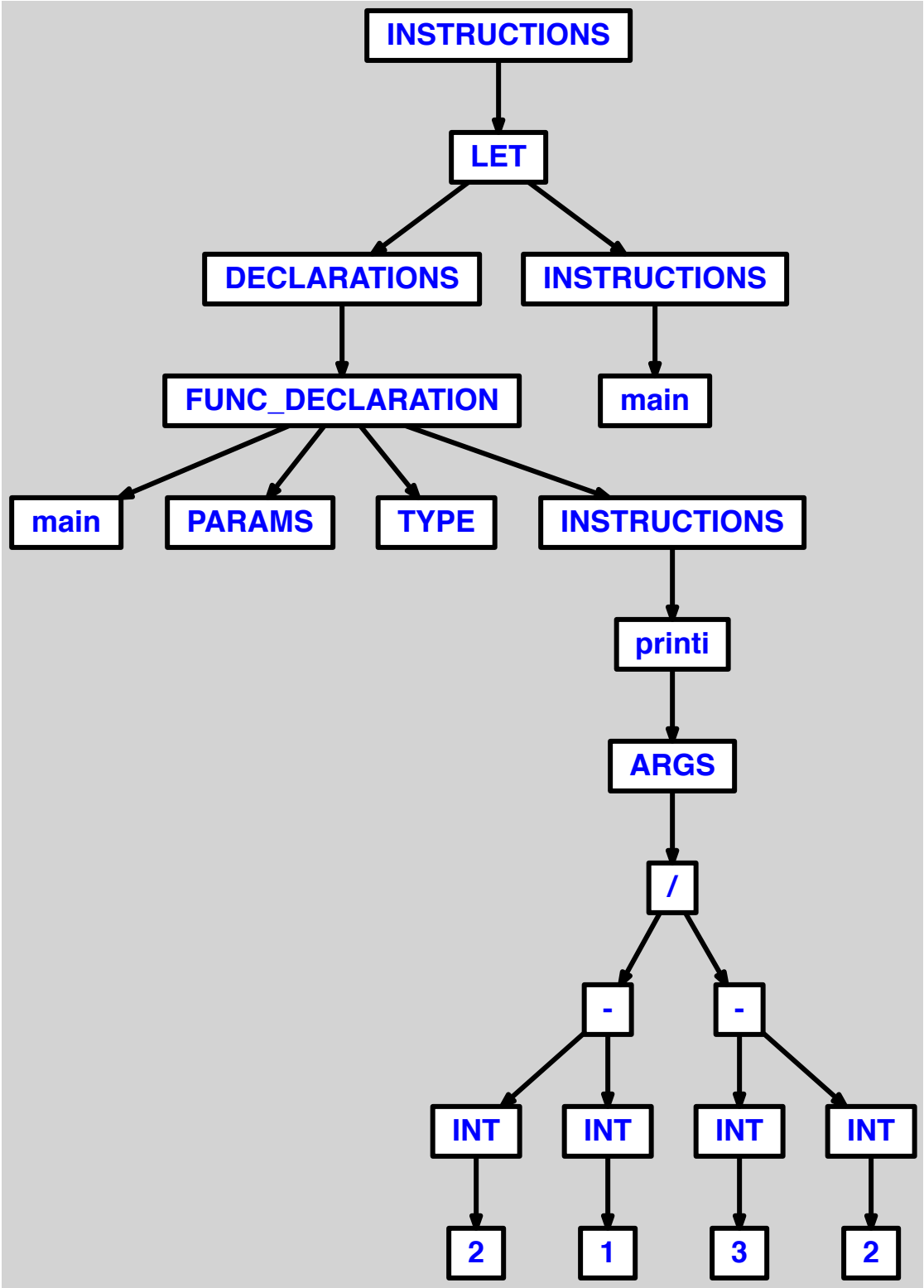
3.2.66 multiplication de 3 additions parenthesees

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


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

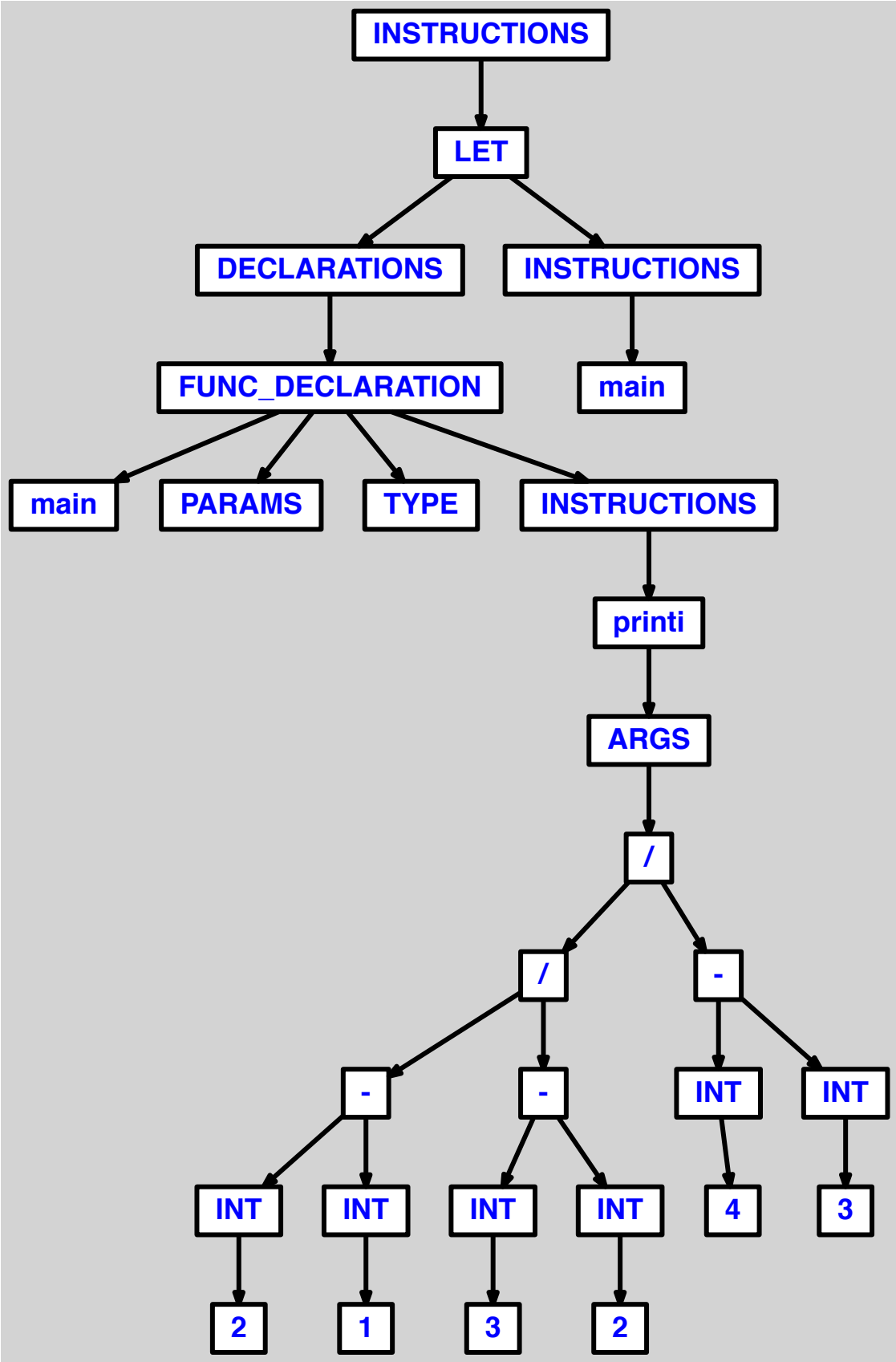
3.2.67 division de 2 soustractions parenthesees

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


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

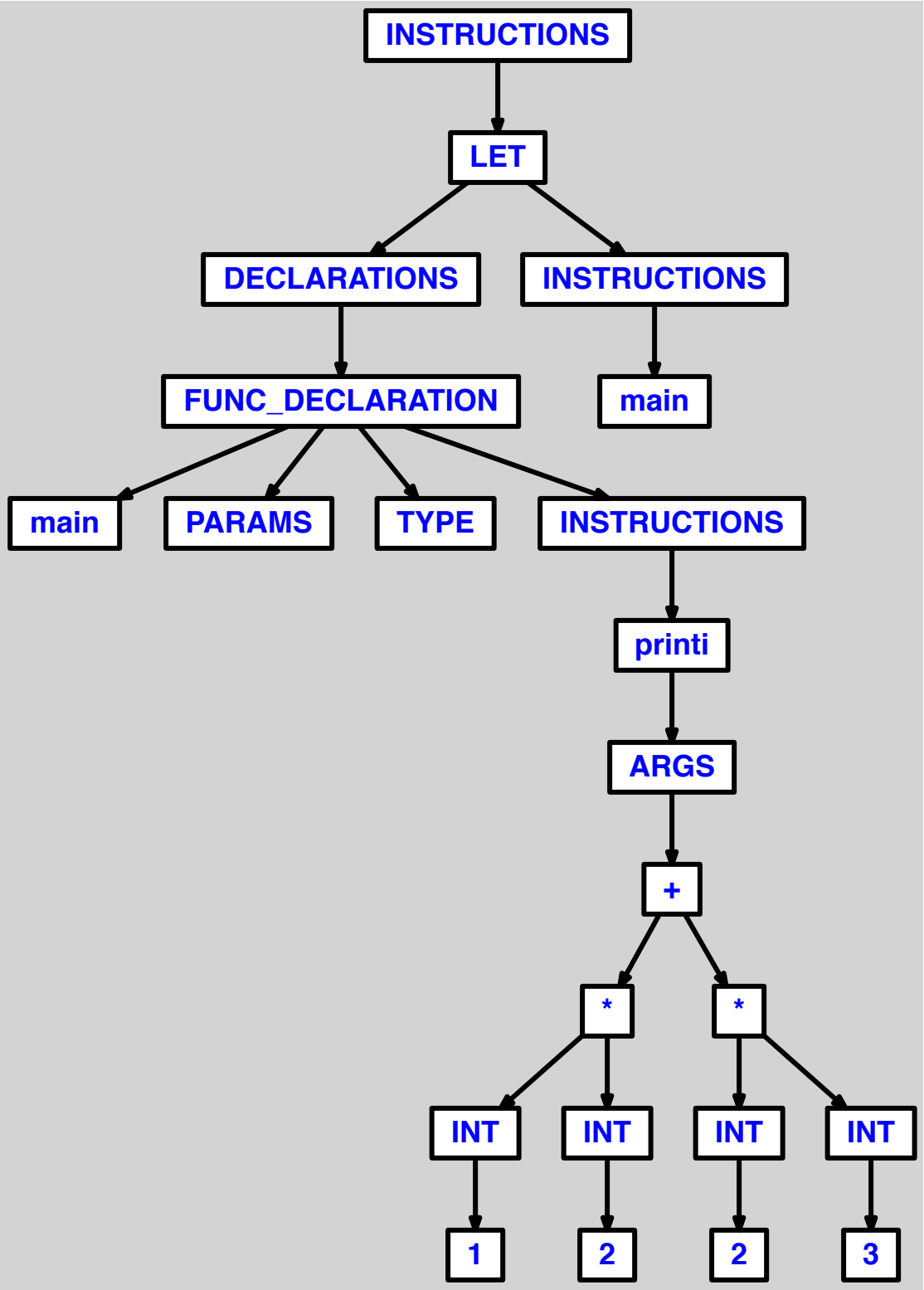
3.2.68 division de 3 soustractions parenthesees

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


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

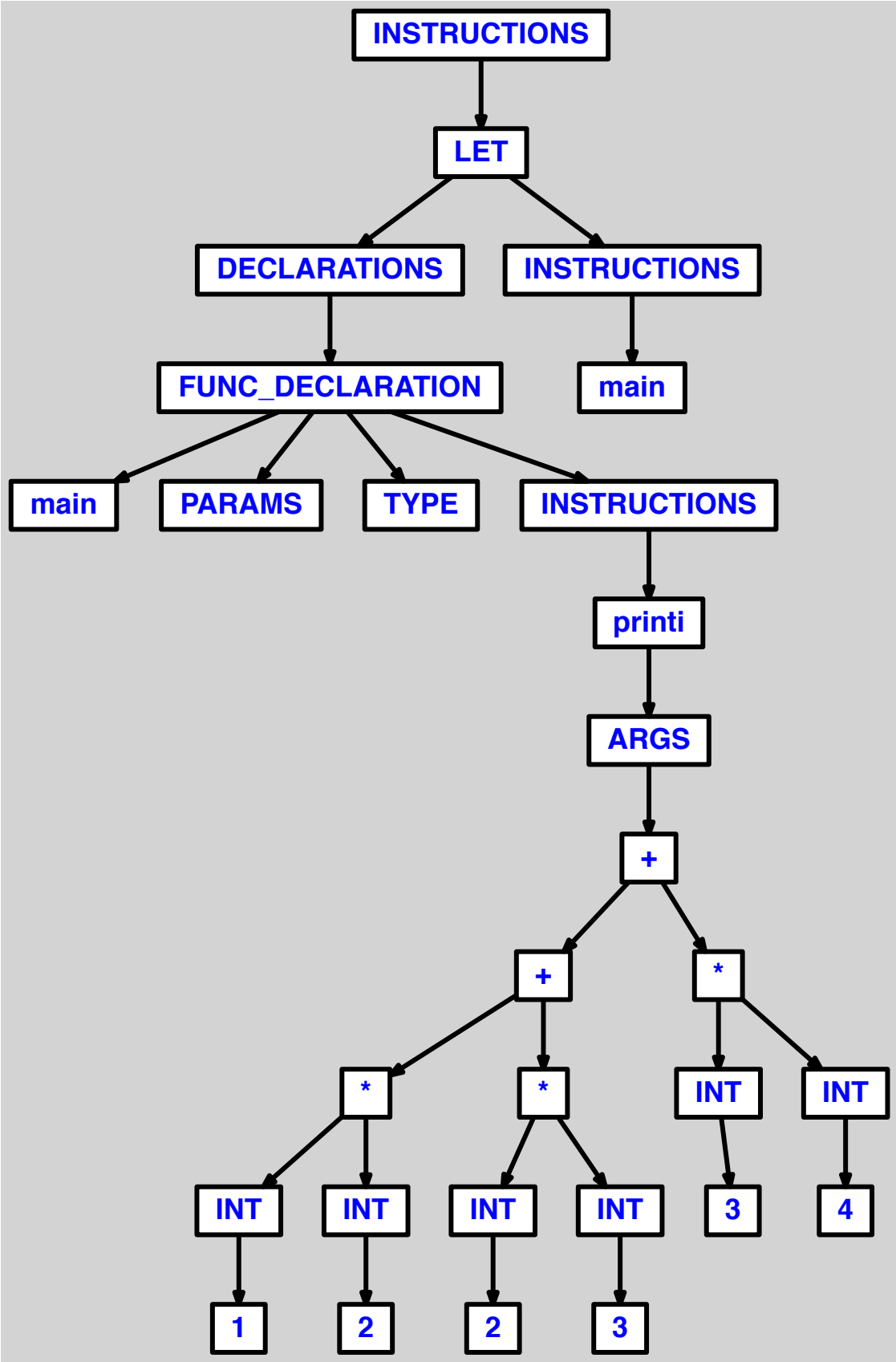
3.2.69 addition de 2 multiplications parenthesees

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


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

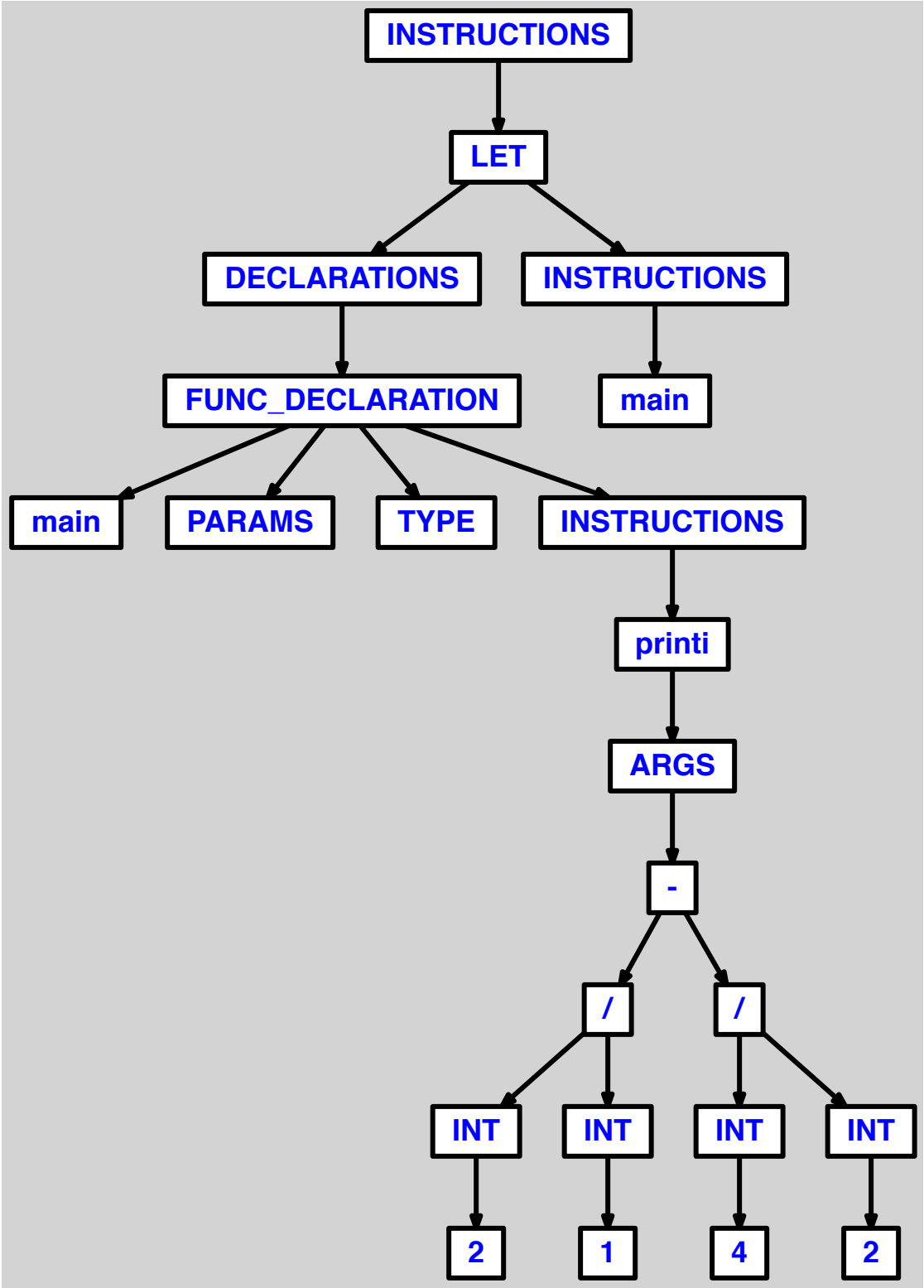
3.2.70 addition de 3 multiplications parenthesees

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


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

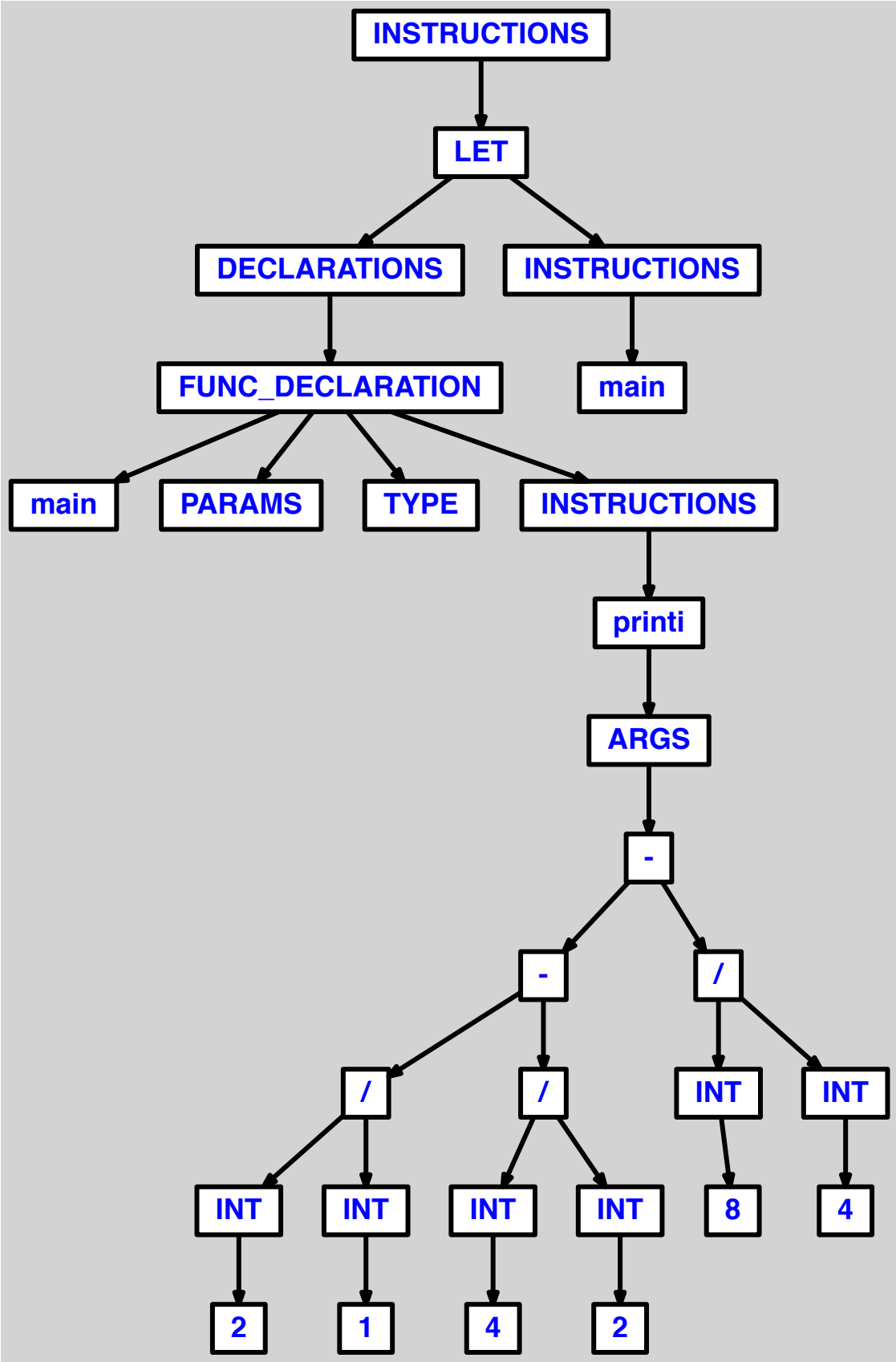
3.2.71 soustraction de 2 divisions parenthesees

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


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

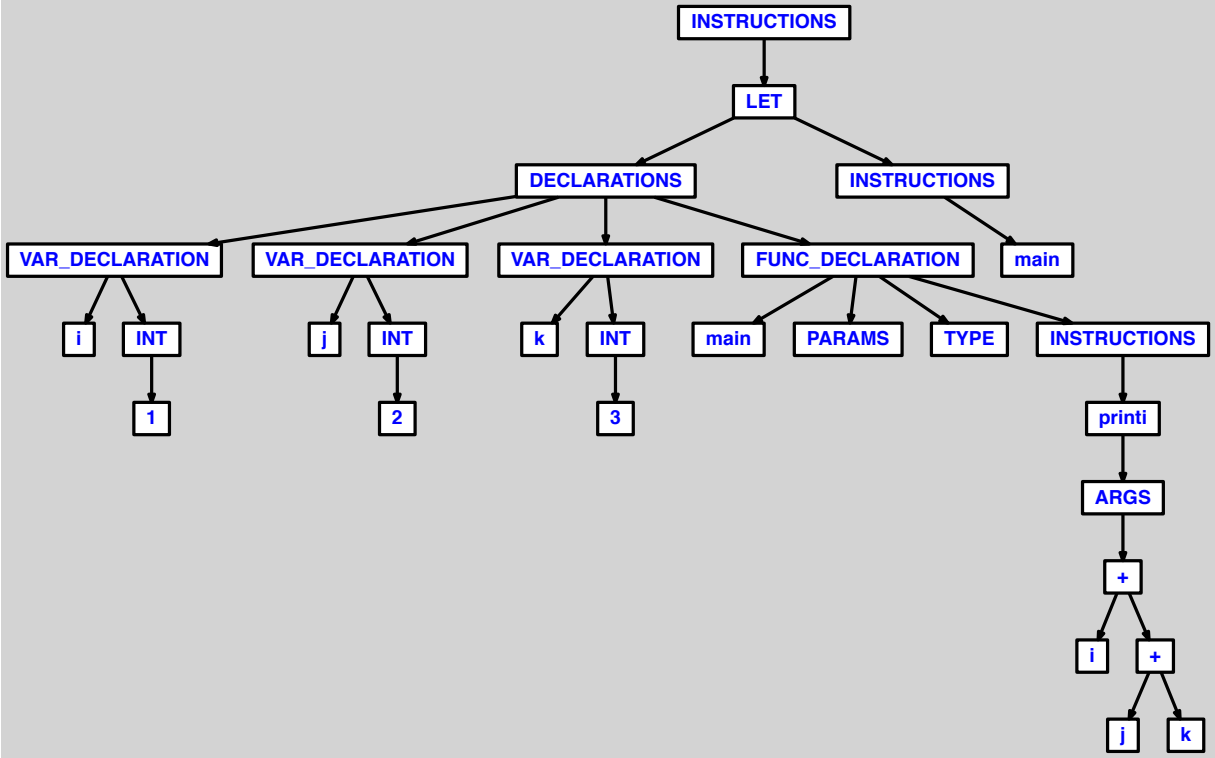
3.2.72 soustraction de 3 divisions parenthesees

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


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

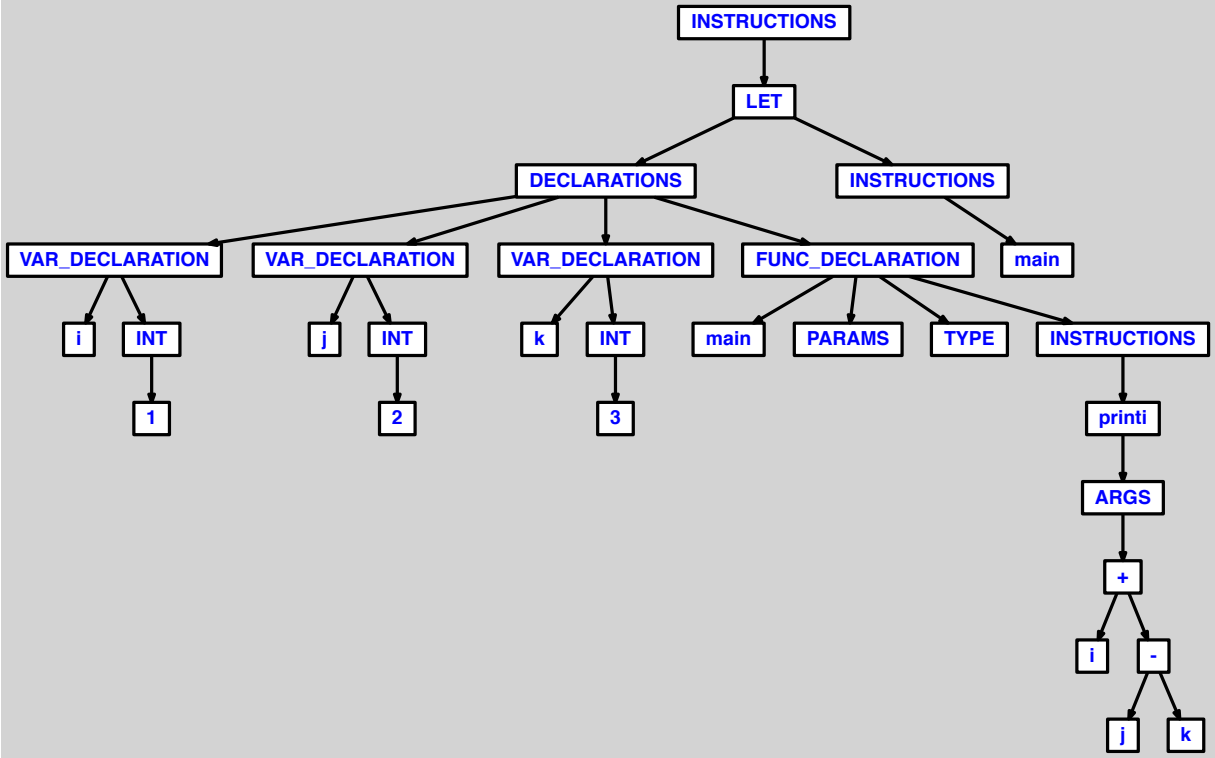
3.2.73 addition a 3 termes identifies par variables, avec parenthesage des 2 variables a droite

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
i
+
j
k

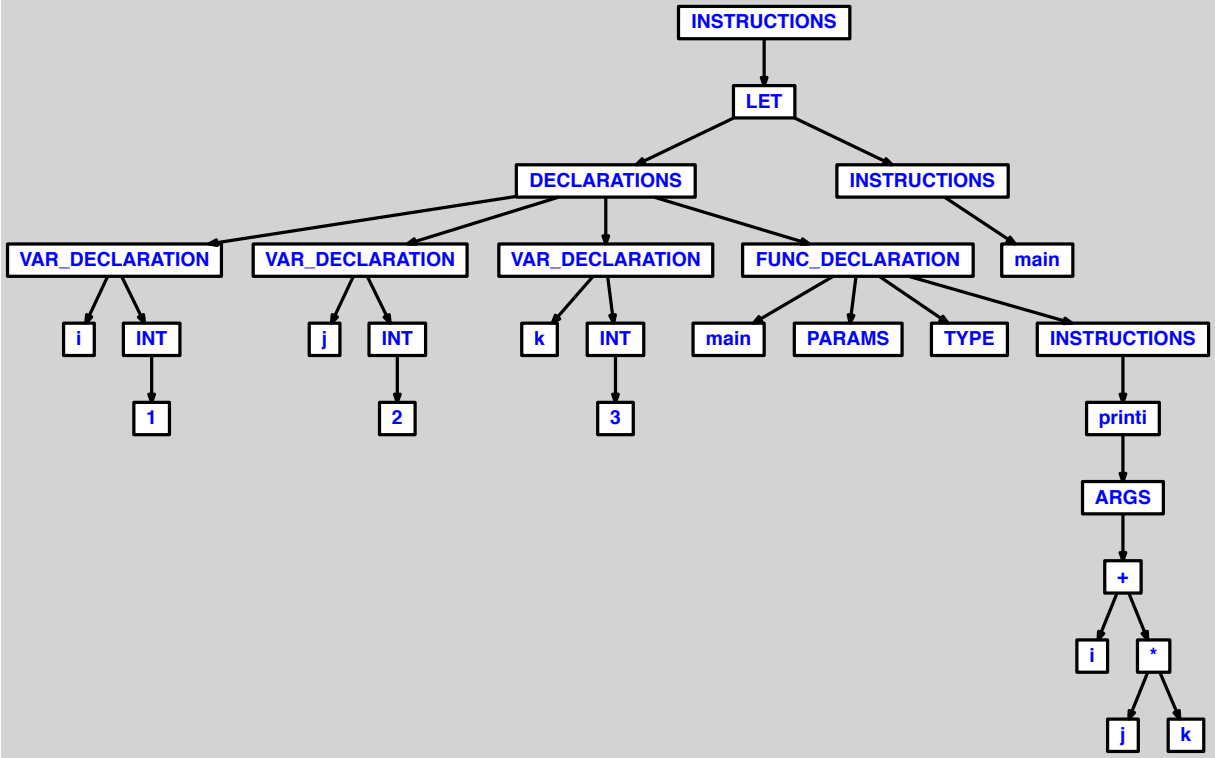
3.2.74 addition suivie de soustraction, avec termes identifies par variables et parenthesage des 2 variables a droite

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
i
-
j
k

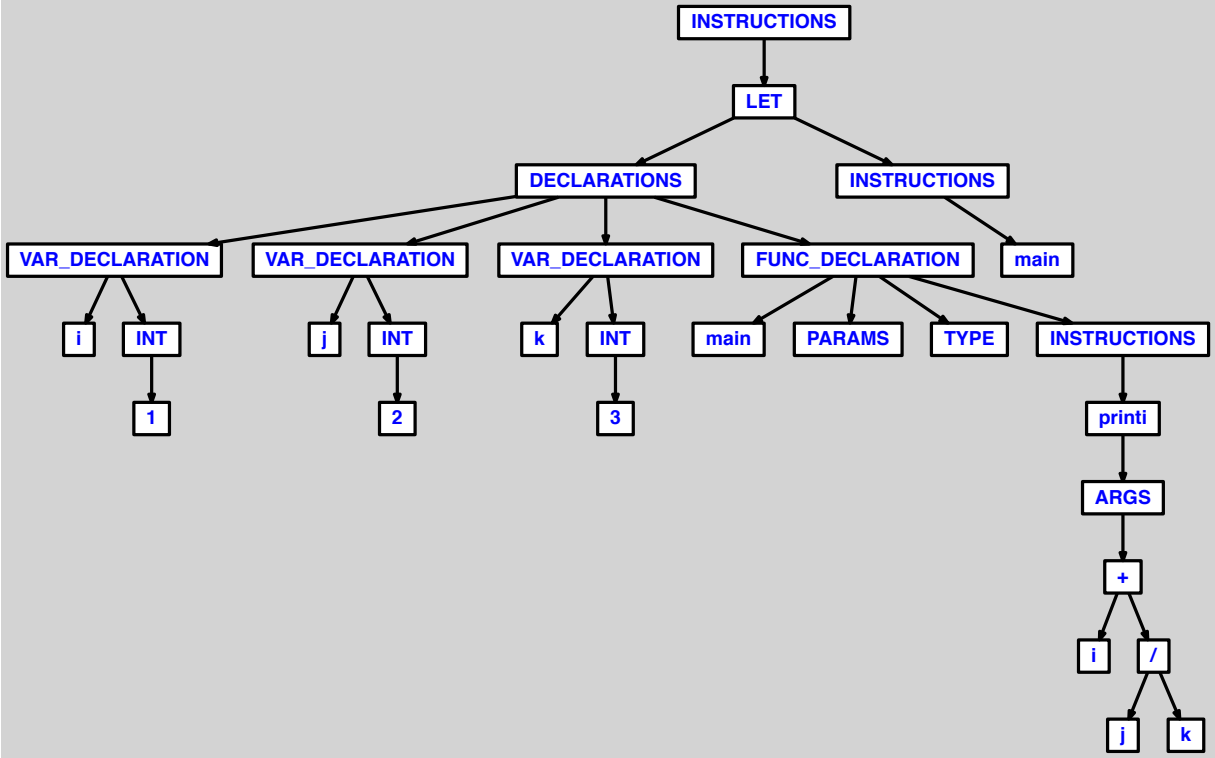
3.2.75 addition suivie de multiplication, avec termes identifies par variables et parenthesage des 2 variables a droite

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
i
*
j
k

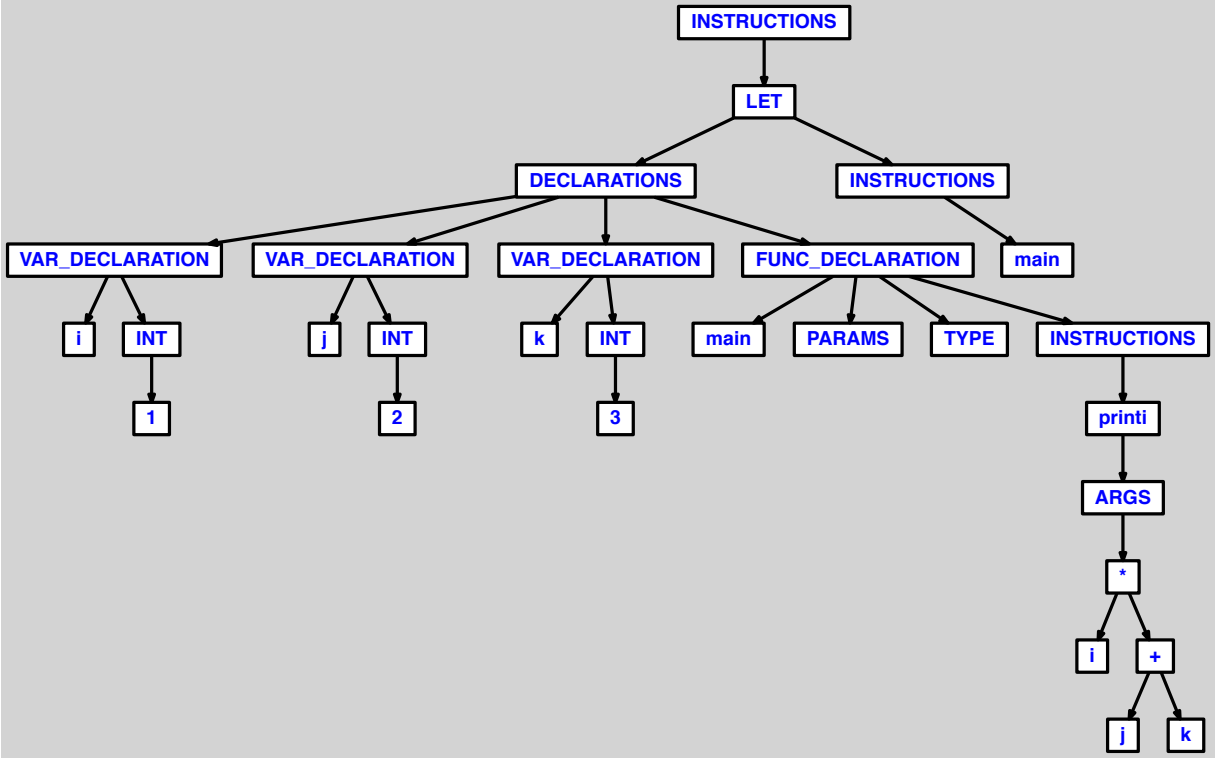
3.2.76 addition suivie de division, avec termes identifies par variables et parenthesage des 2 variables a droite

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
i
/
j
k

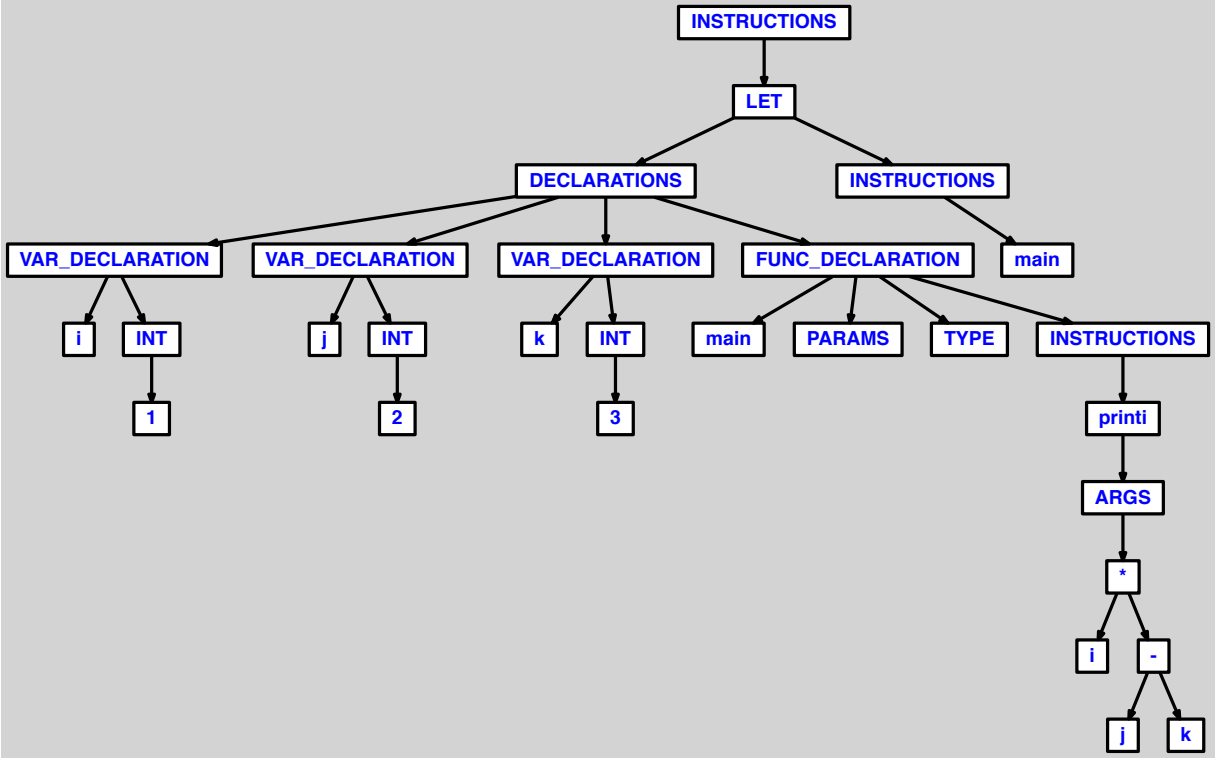
3.2.77 multiplication suivie d'addition, avec termes identifies par variables et parenthesage des 2 variables a droite

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
*
i
+
j
k

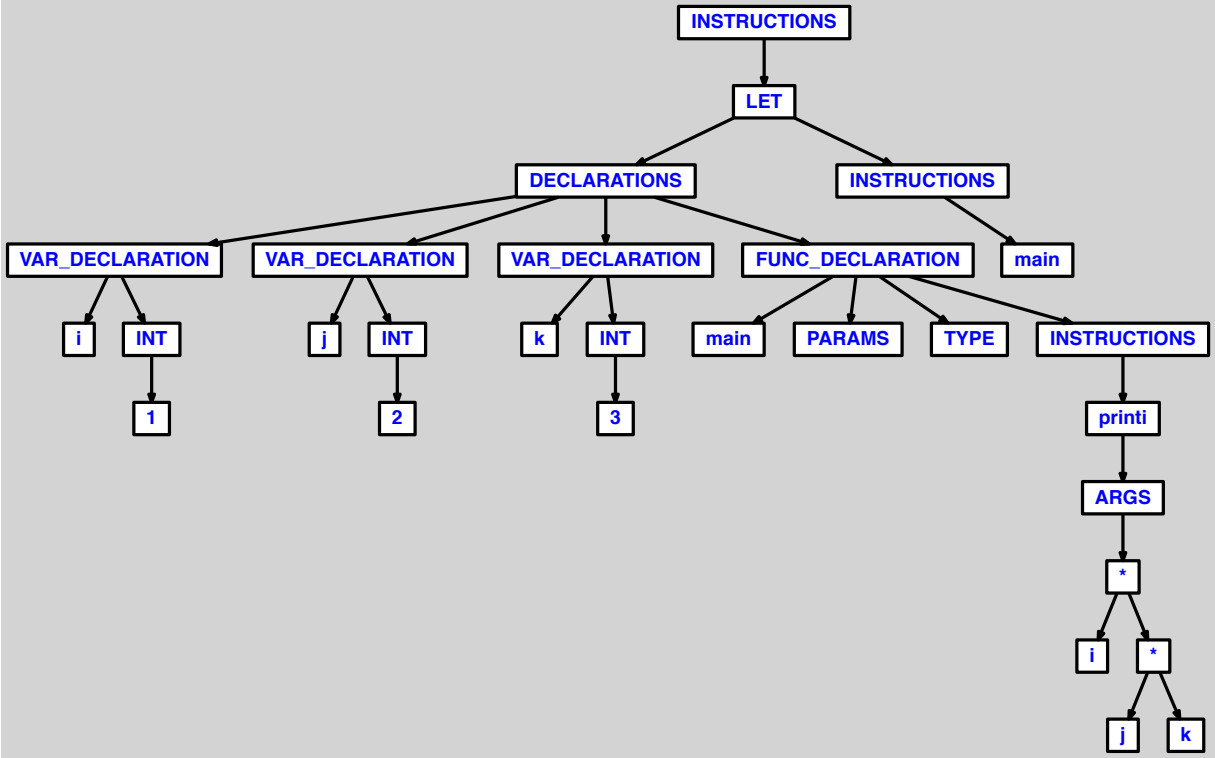
3.2.78 multiplication suivie de soustraction, avec termes identifies par variables et parenthesage des 2 variables a droite

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
*
i
-
j
k

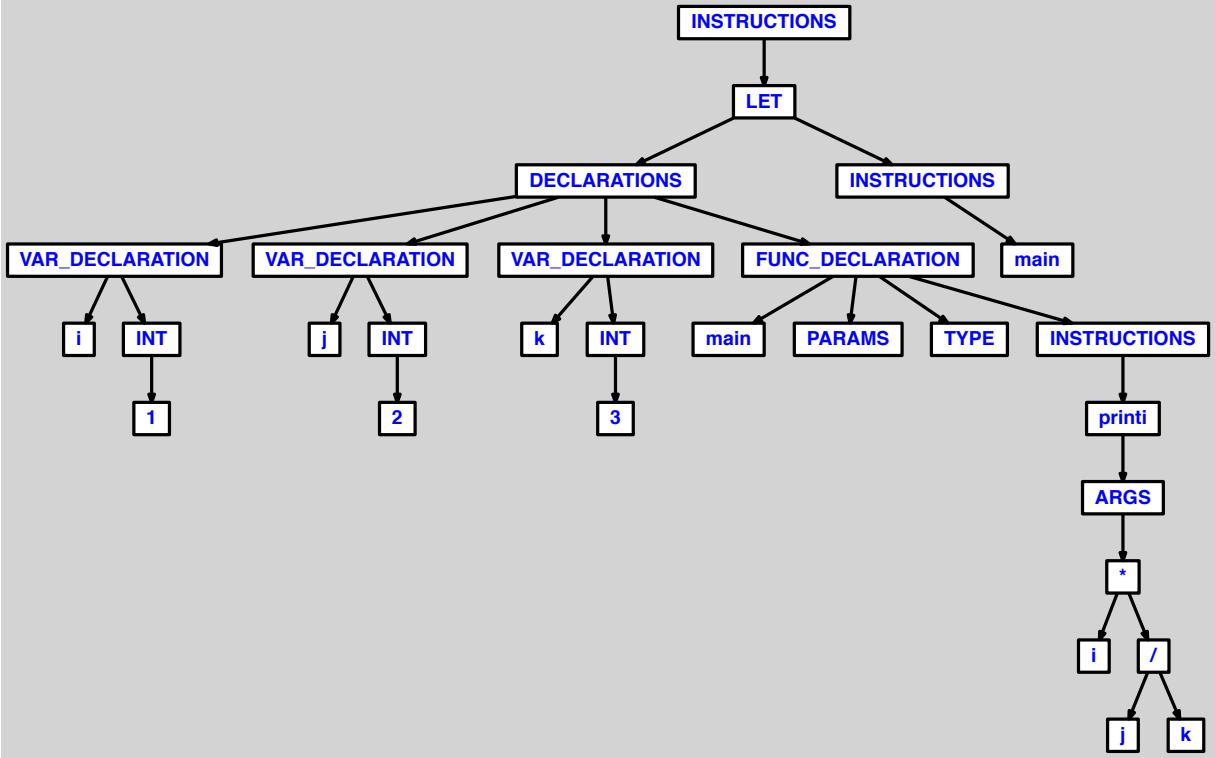
3.2.79 multiplication a 3 termes identifies par variables, avec parenthesage des 2 variables a droite

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
*
i
*
j
k

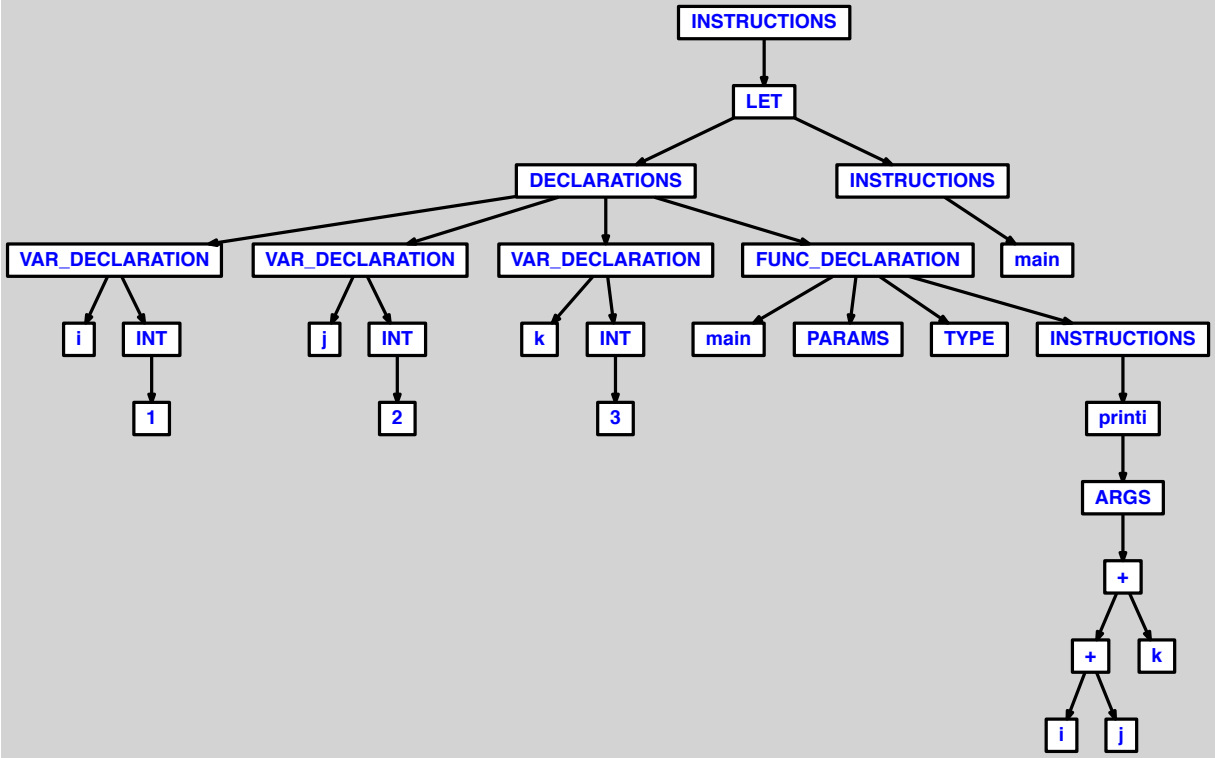
3.2.80 multiplication suivie de division, avec termes identifies par variables et parenthesage des 2 variables a droite

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
*
i
/
j
k

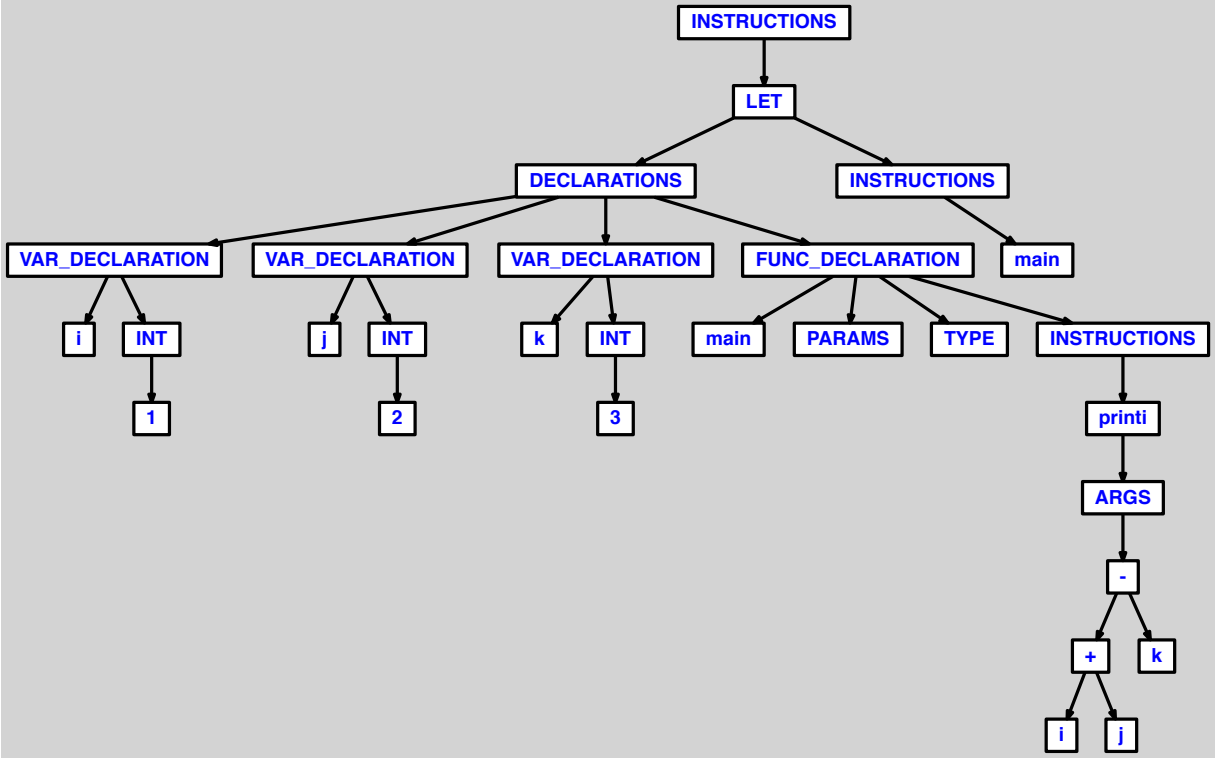
3.2.81 addition a 3 termes identifies par variables, avec parenthesage des 2 variables a gauche

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
+
k
i
j

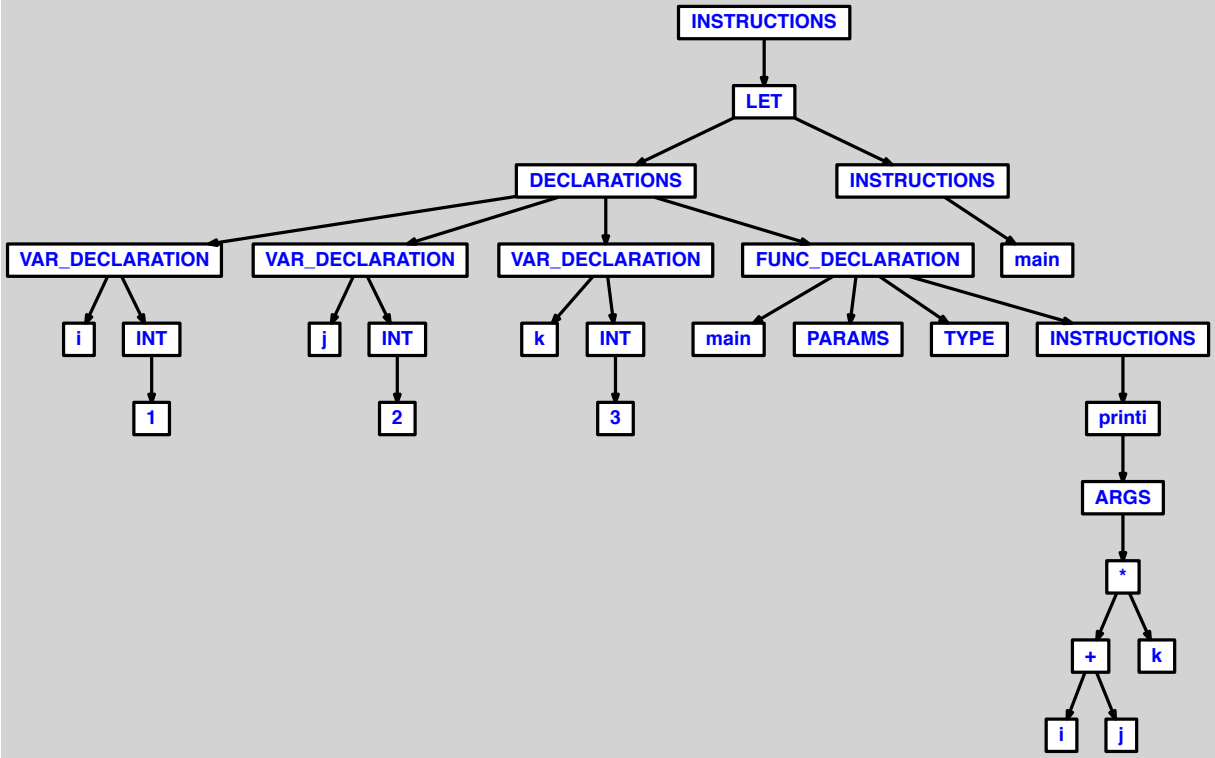
3.2.82 addition suivie de soustraction, avec termes identifies par variables et parenthesage des 2 variables a gauche

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
-
+
k
i
j

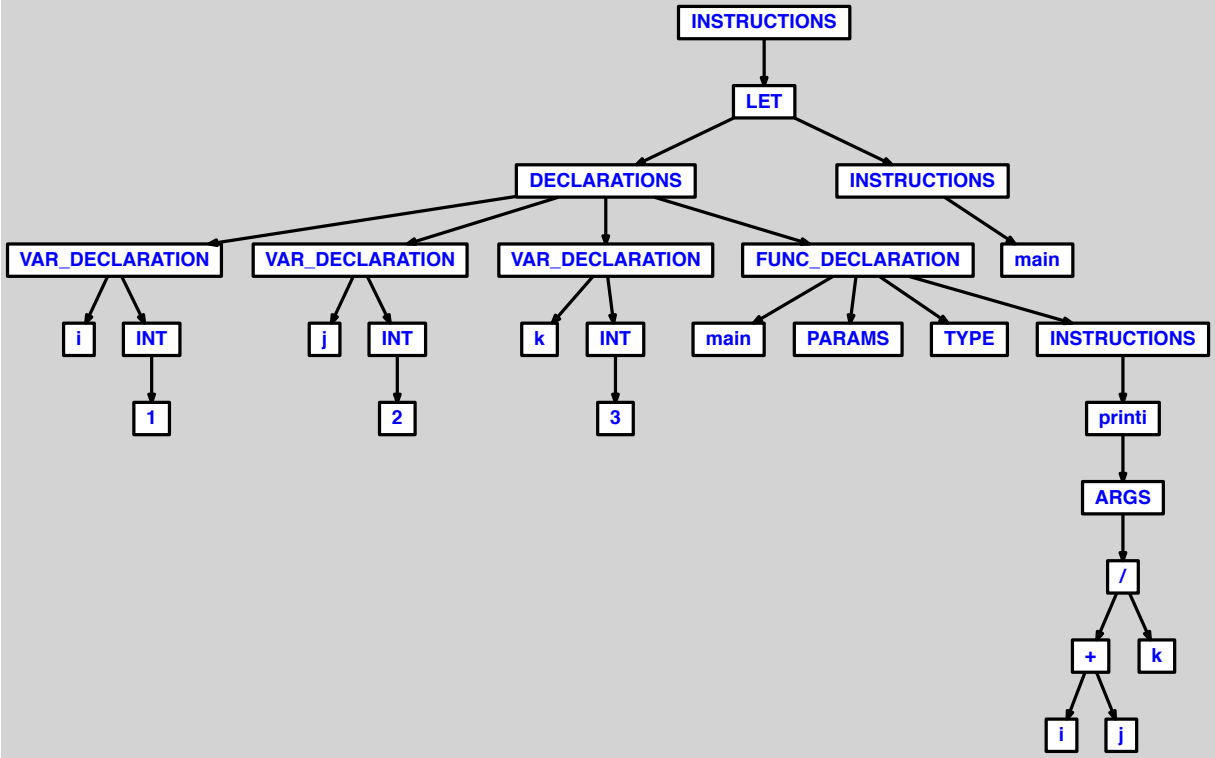
3.2.83 addition suivie de multiplication, avec termes identifies par variables et parenthesage des 2 variables a gauche

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
*
+
k
i
j

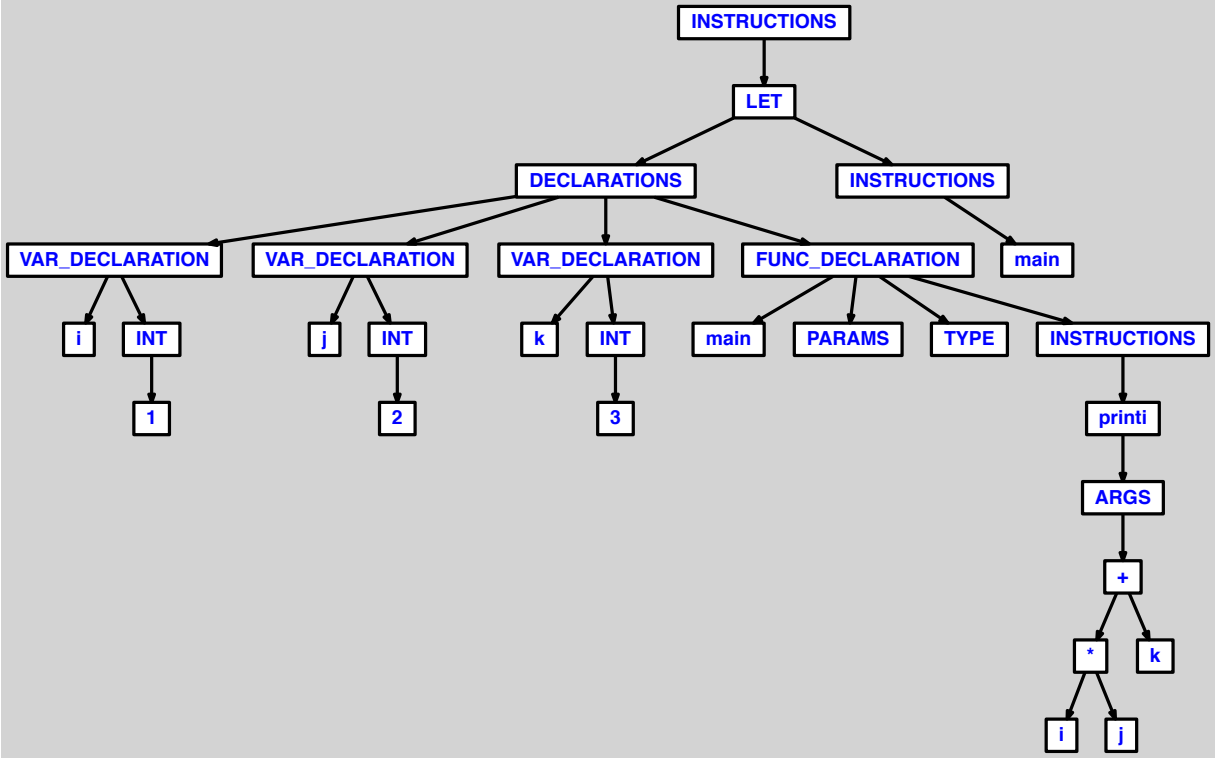
3.2.84 addition suivie de division, avec termes identifies par variables et parenthesage des 2 variables a gauche

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
/
+
k
i
j

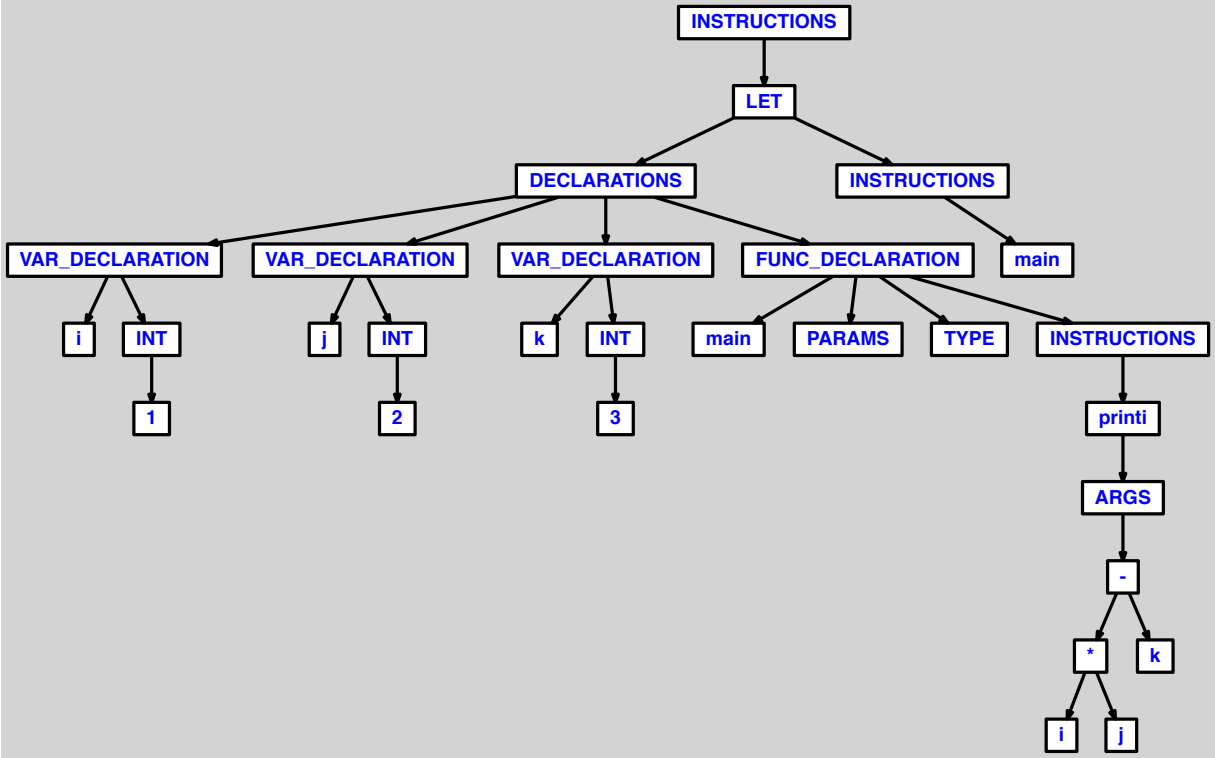
3.2.85 multiplication suivie d'addition, avec termes identifies par variables et parenthesage des 2 variables a gauche

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
*
k
i
j

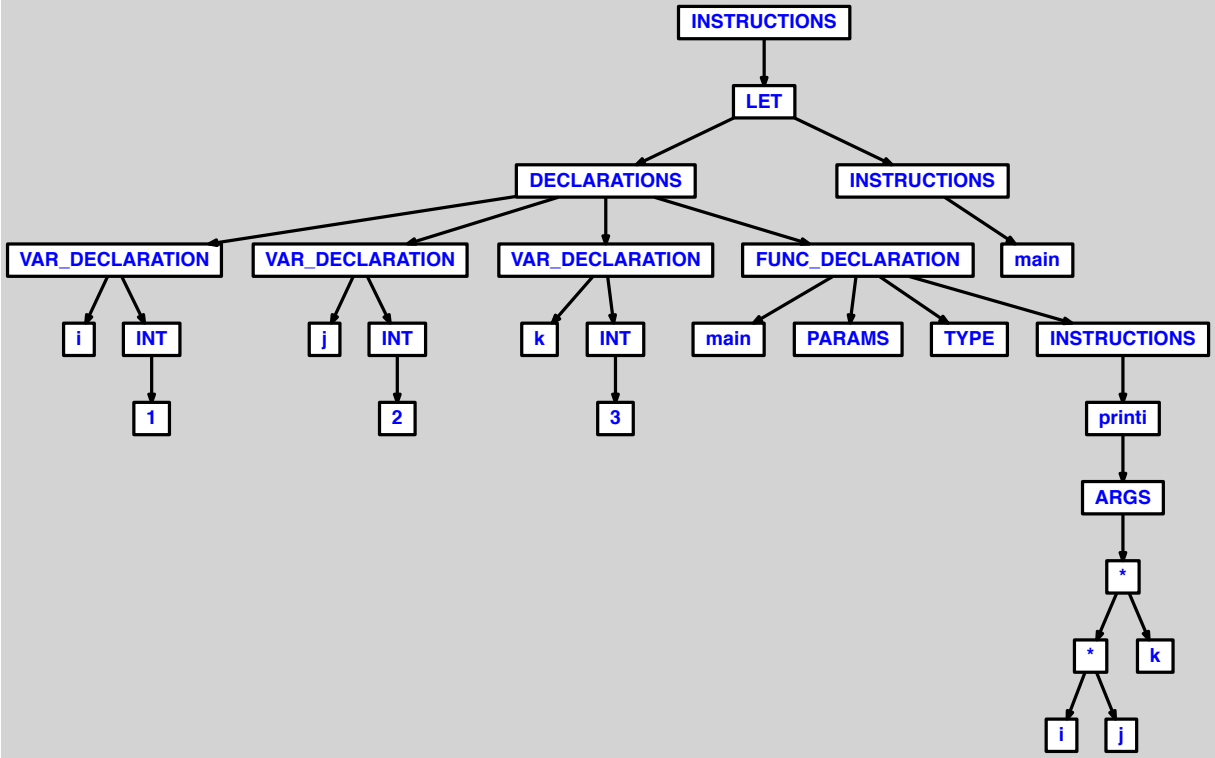
3.2.86 multiplication suivie de soustraction, avec termes identifies par variables et parenthesage des 2 variables a gauche

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
-
*
k
i
j

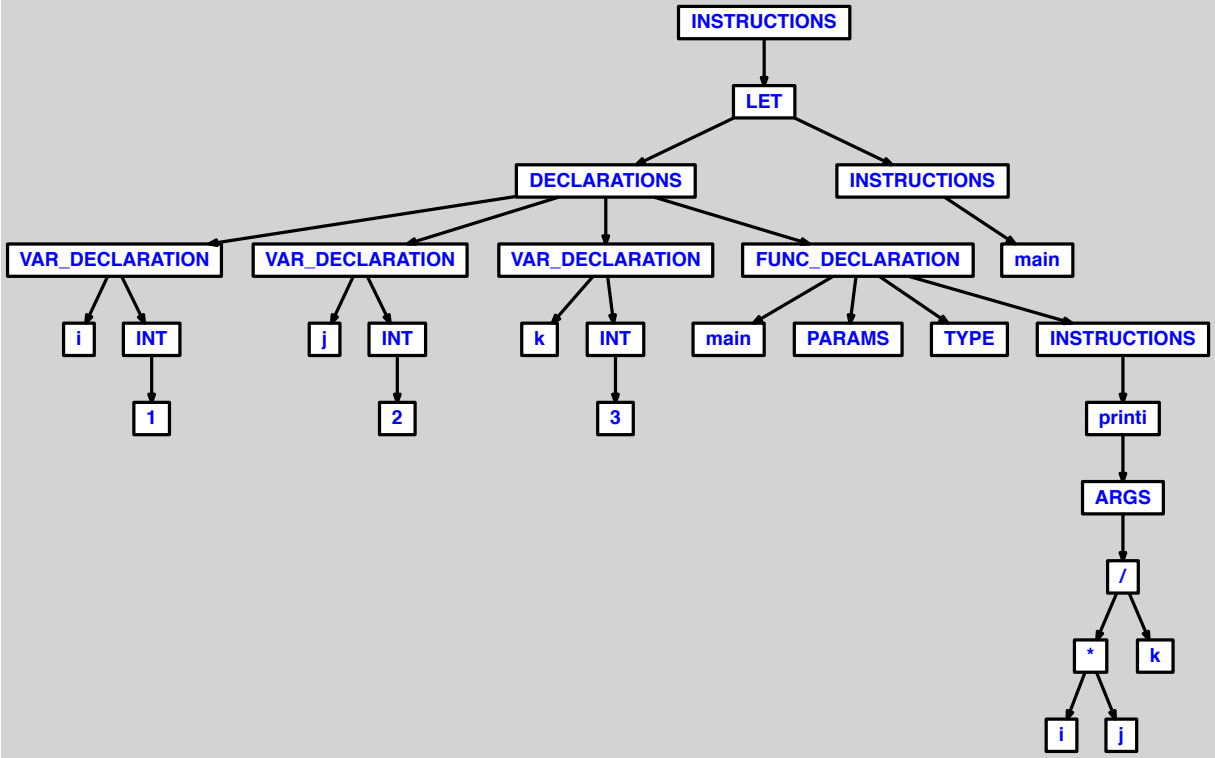
3.2.87 multiplication a 3 termes identifies par variables, avec parenthesage des 2 variables a gauche

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
*
*
k
i
j

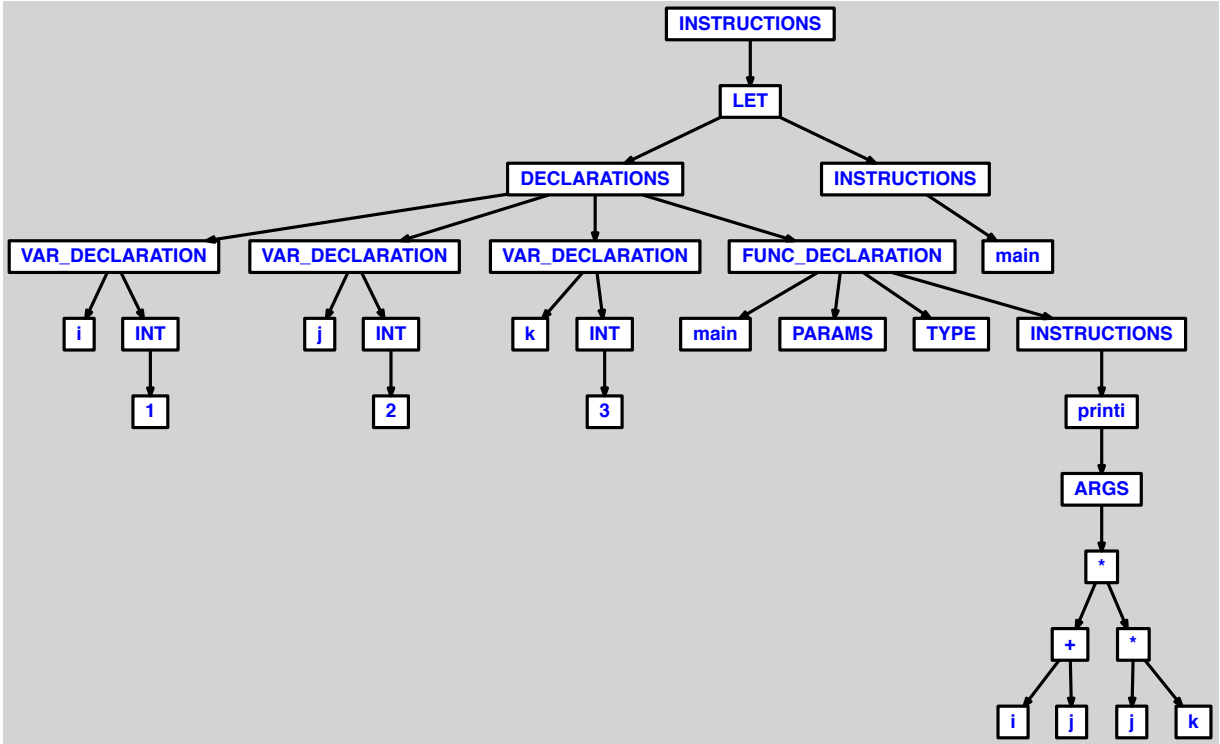
3.2.88 multiplication suivie de division, avec termes identifies par variables et parenthesage des 2 variables a gauche

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
/
*
k
i
j

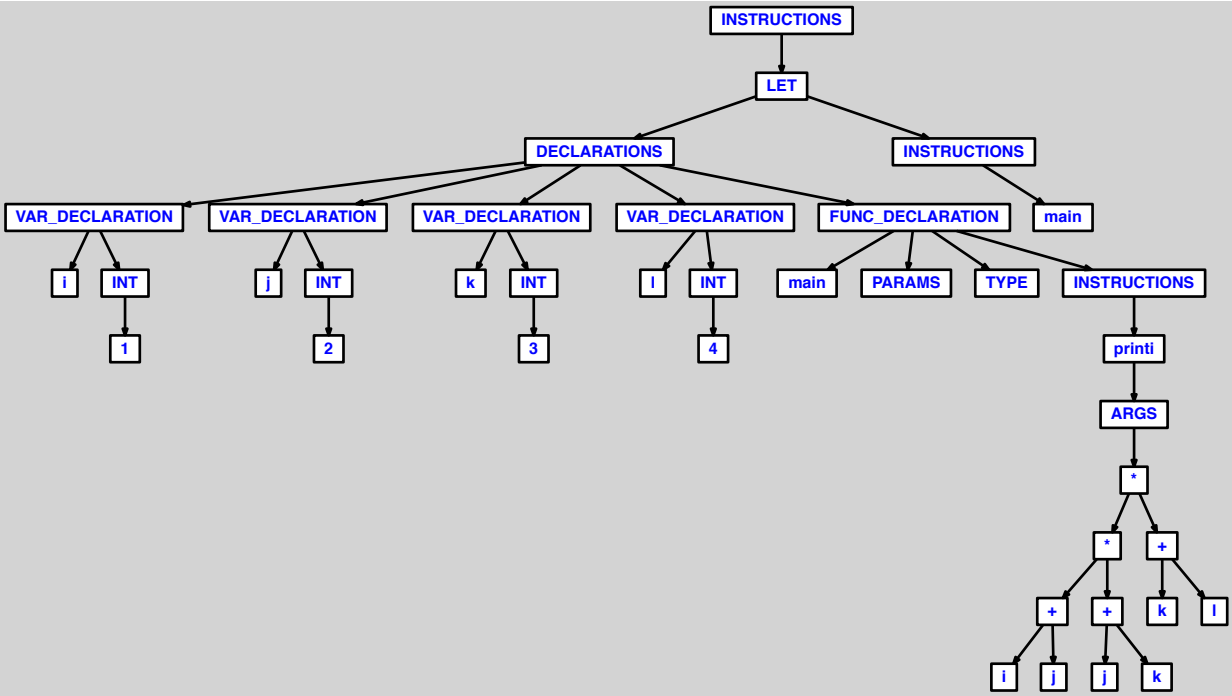
3.2.89 multiplication de 2 additions parenthesees, avec termes identifies par variables

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
*
+
*
i
j
j
k

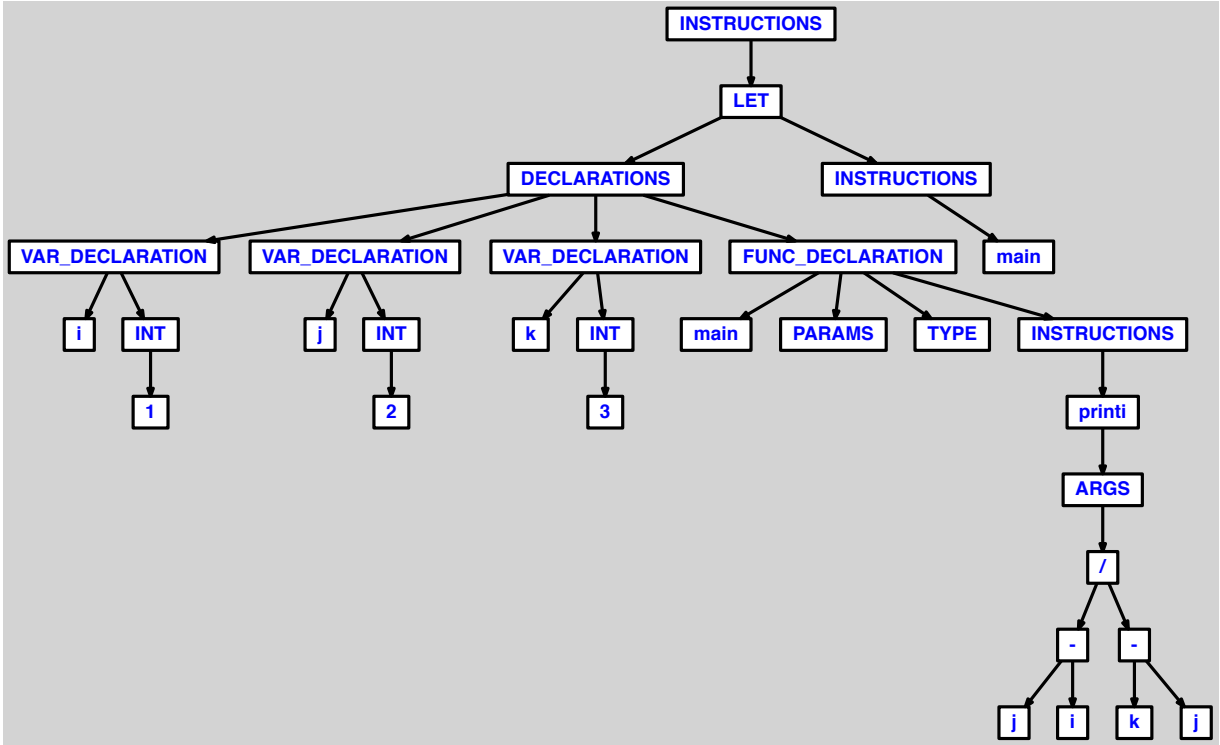
3.2.90 multiplication de 3 additions parenthesees, avec termes identifies par variables

```
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
main
i
INT
j
INT
k
INT
l
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
4
printi
ARGS
*
*
+
+
+
k
l
i
j
j
k

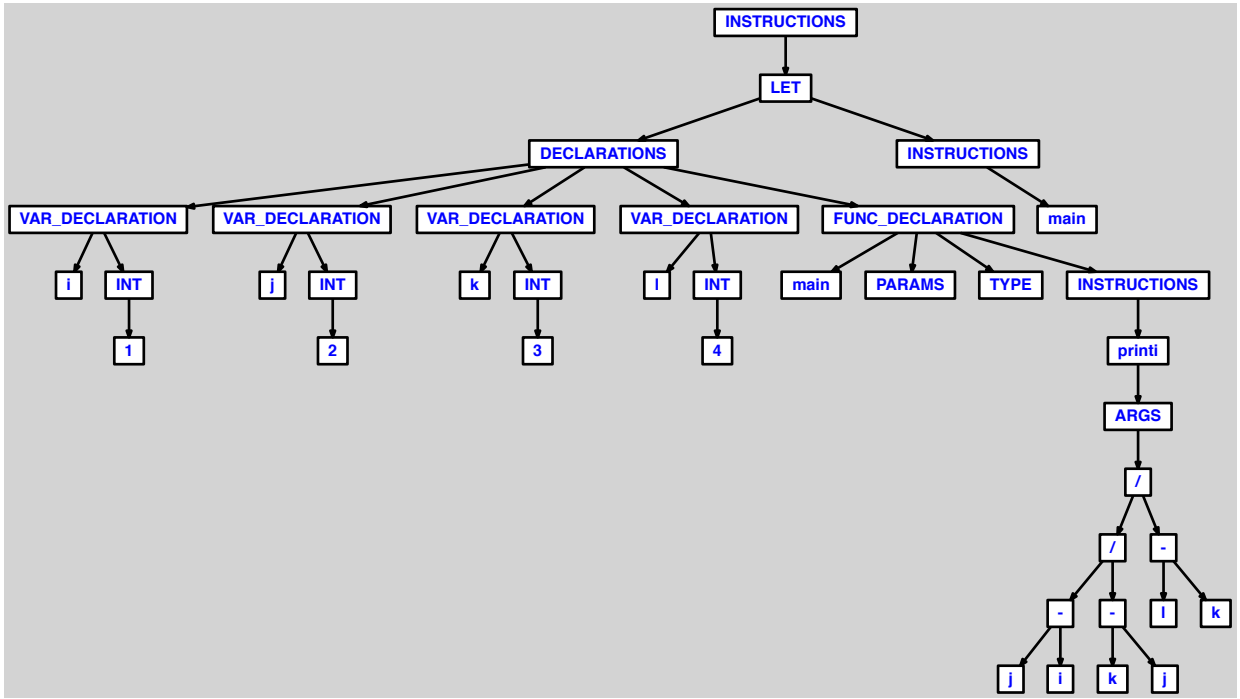
3.2.91 division de 2 soustractions parenthesees, avec termes identifies par variables

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
/
-
-
j
i
k
j

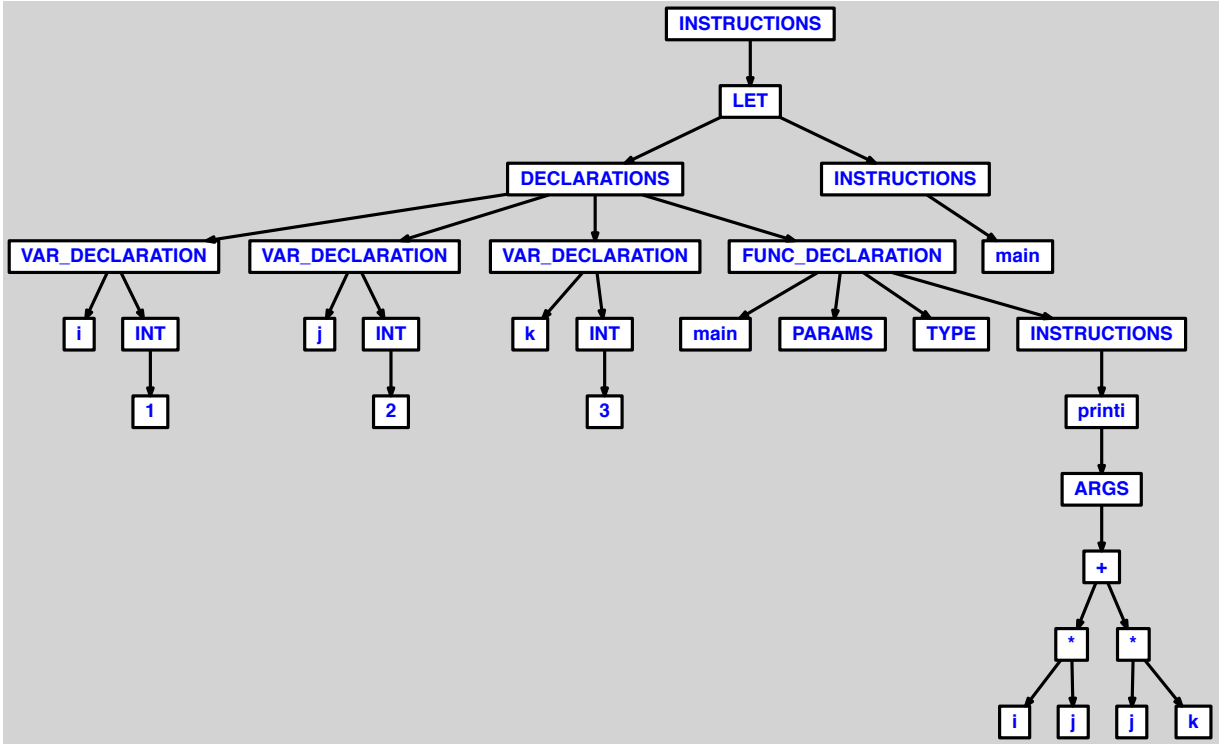
3.2.92 division de 3 soustractions parenthesees, avec termes identifies par variables

```
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
main
i
INT
1
j
INT
2
k
INT
3
l
INT
4
main
PARAMS
TYPE
INSTRUCTIONS
printi
ARGS
/
/
-
-
-
l
k
j
i
k
j

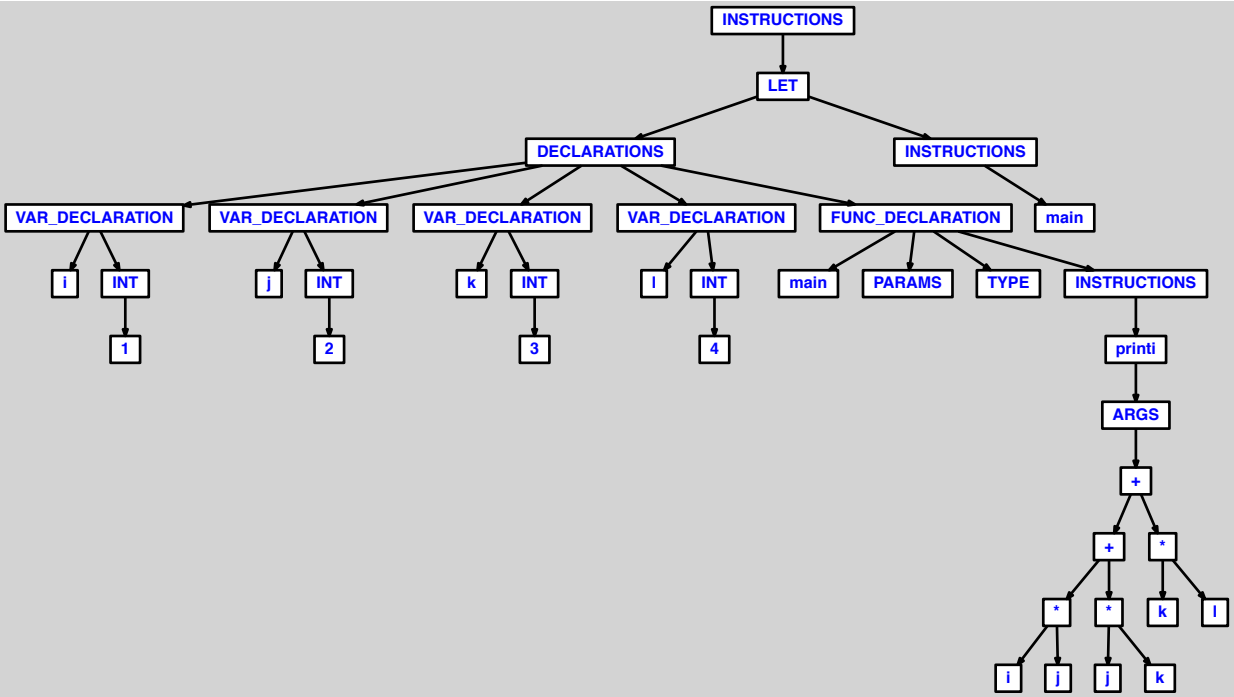
3.2.93 addition de 2 multiplications parenthesees, avec termes identifies par variables

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
printi
ARGS
+
*
*
i
j
j
k

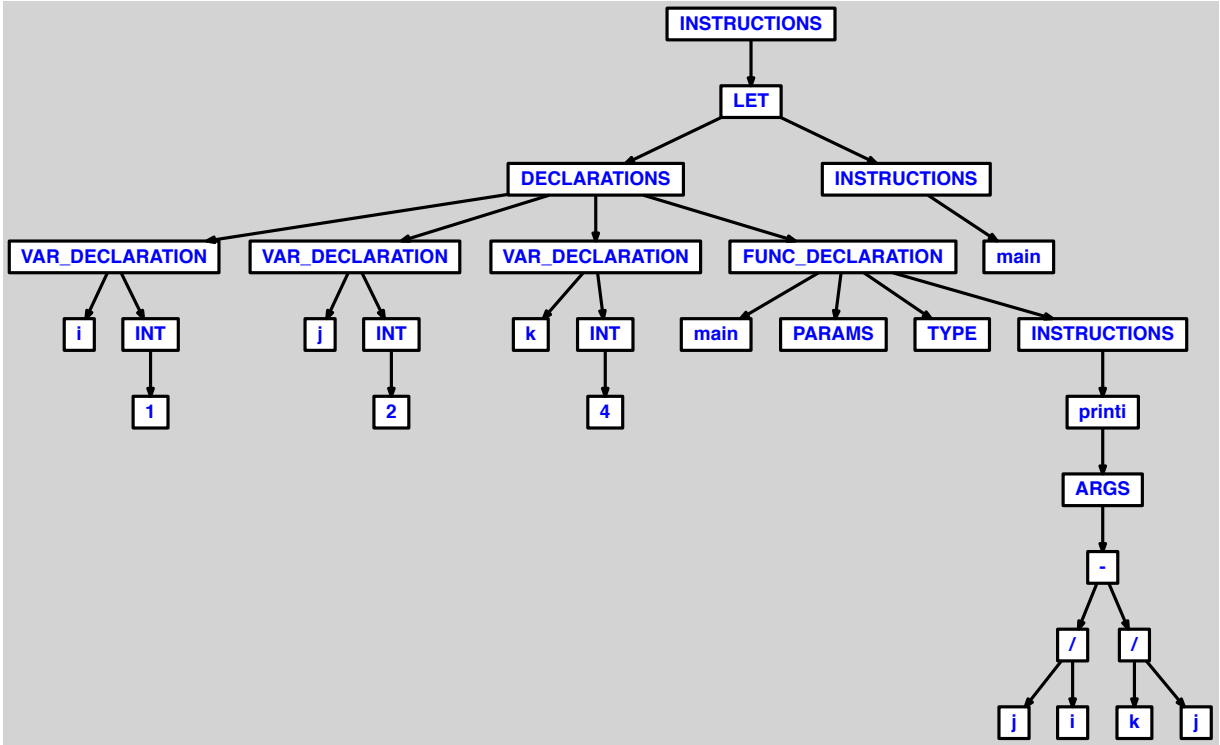
3.2.94 addition de 3 multiplications parenthesees, avec termes identifies par variables

```
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
main
i
INT
j
INT
k
INT
l
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
3
4
printi
ARGS
+
+
*
*
*
k
l
i
j
j
k

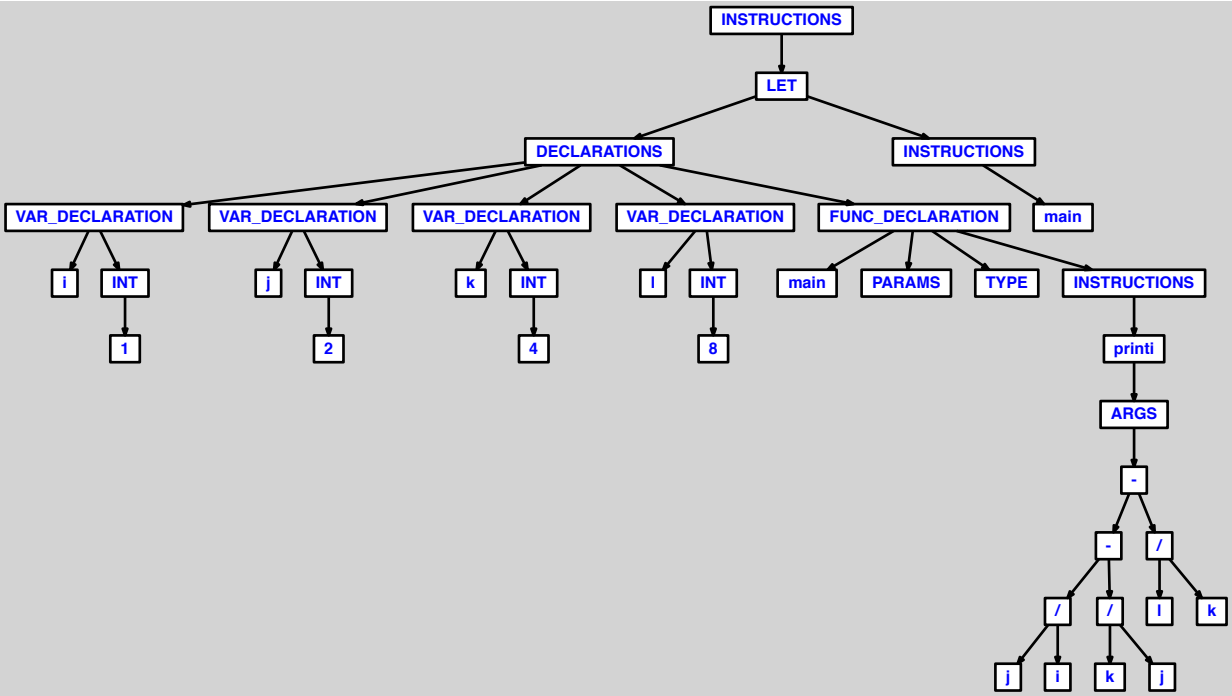
3.2.95 soustraction de 2 divisions parenthesees, avec termes identifies par variables

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
4
printi
ARGS
-
/
/
j
i
k
j

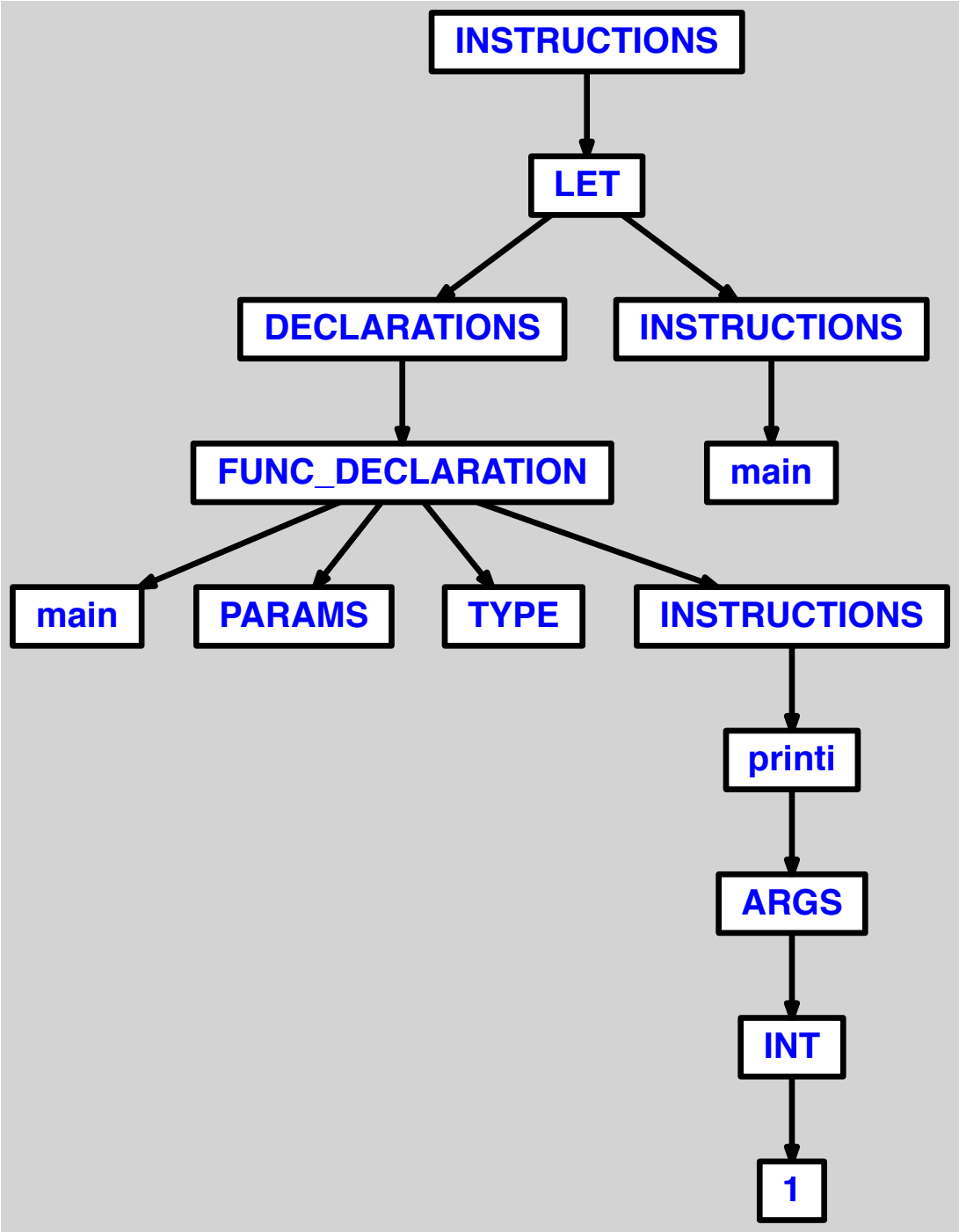
3.2.96 soustraction de 3 divisions parenthesees, avec termes identifies par variables

```
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
main
i
INT
j
INT
k
INT
l
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
2
4
8
printi
ARGS
-
-
/
/
/
l
k
j
i
k
j

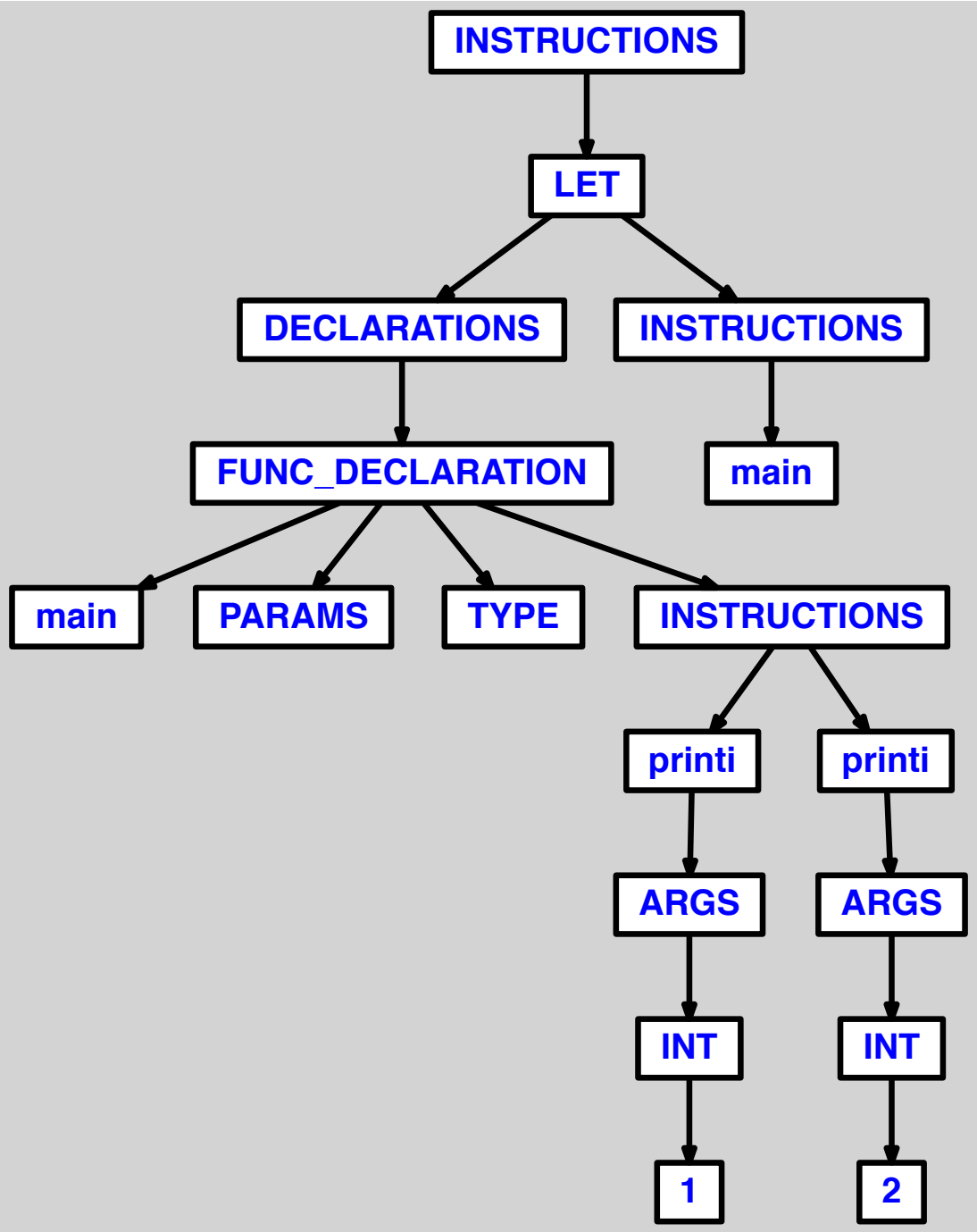
3.2.97 parenthesage d'une instruction

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


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

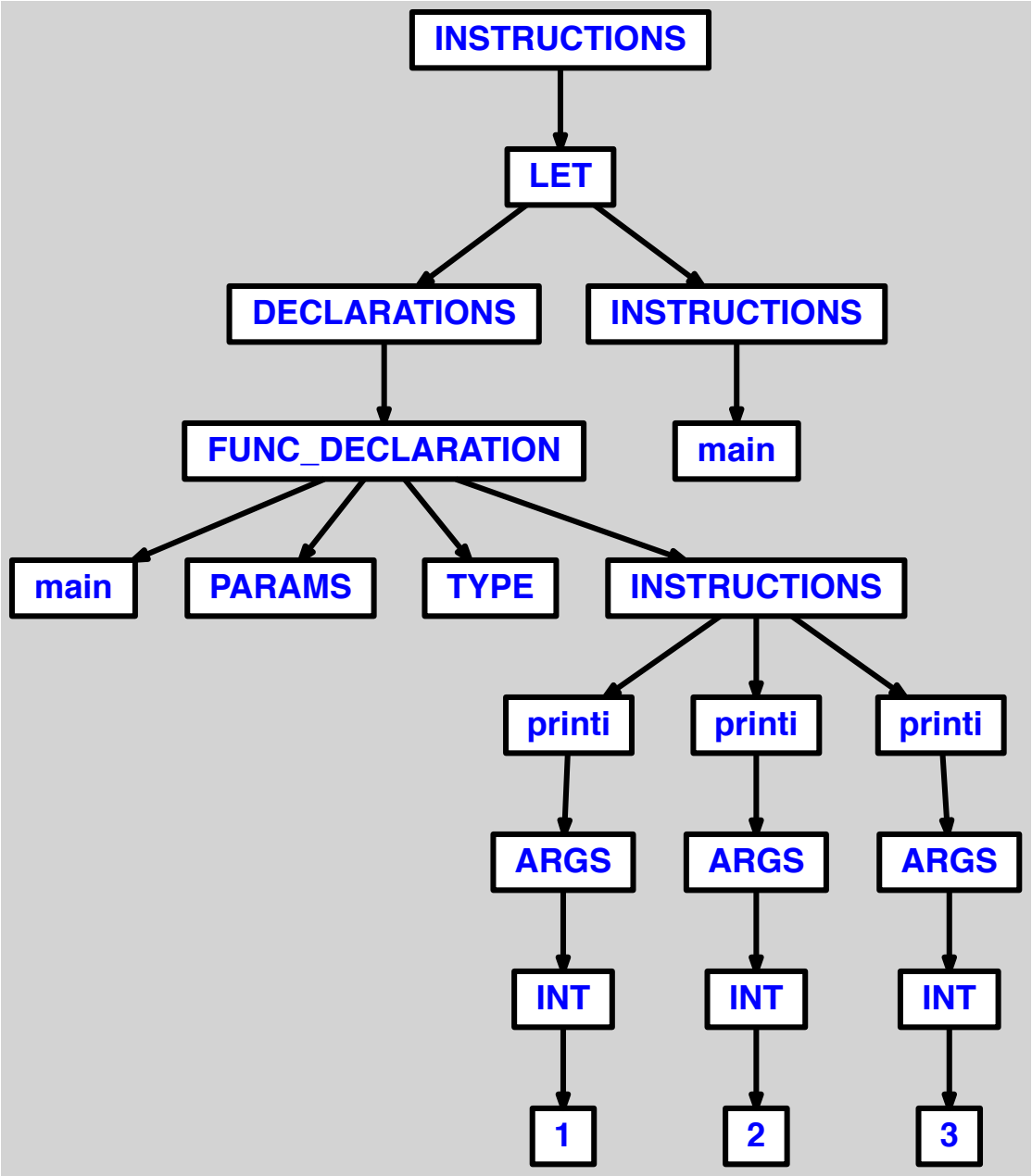
3.2.98 parenthesage de 2 instructions

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


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

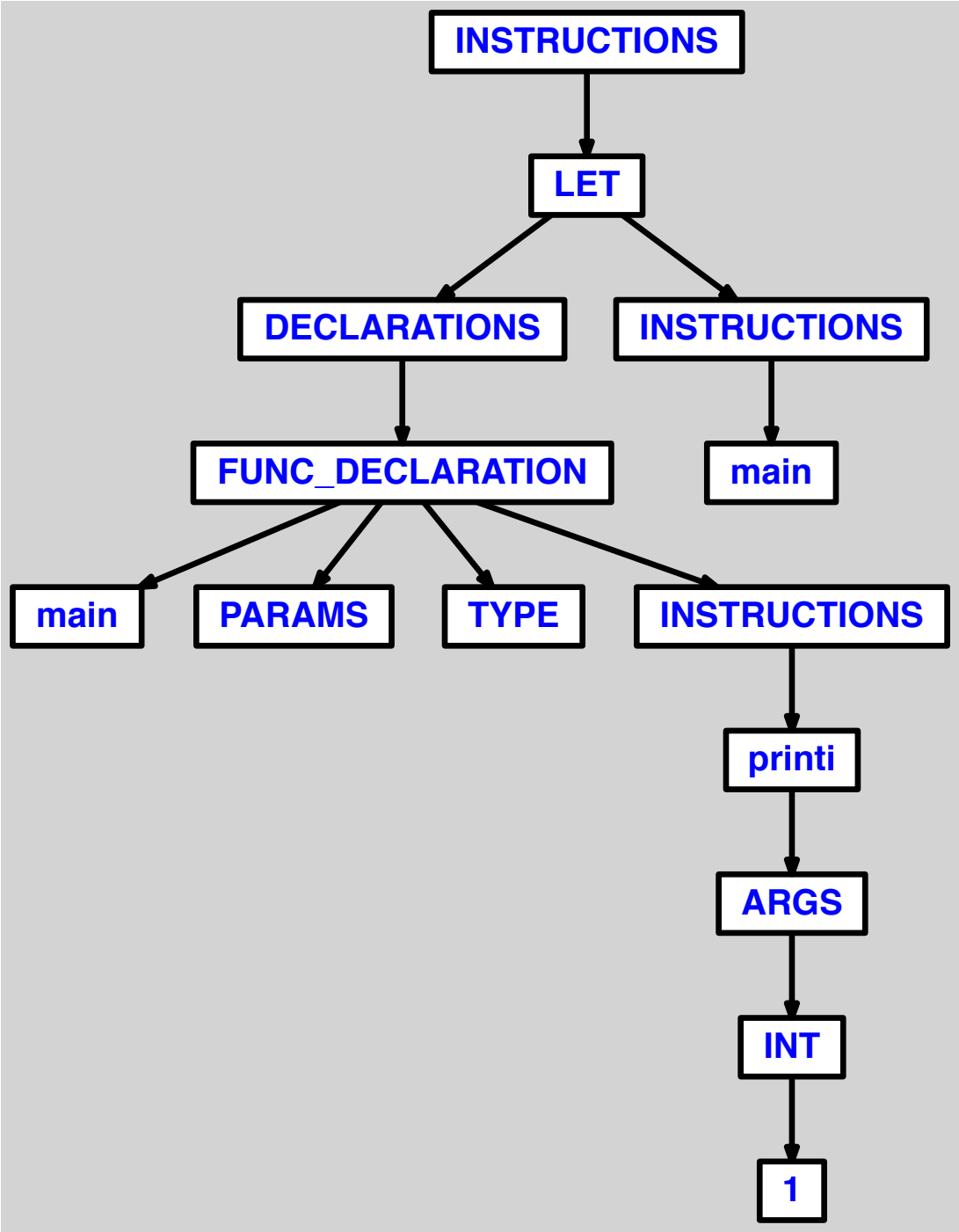
3.2.99 parenthesage de 3 instructions

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


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

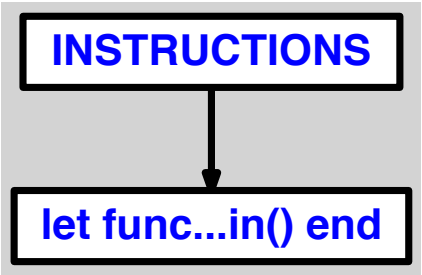
3.2.100 1 instruction sans parentheses

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


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

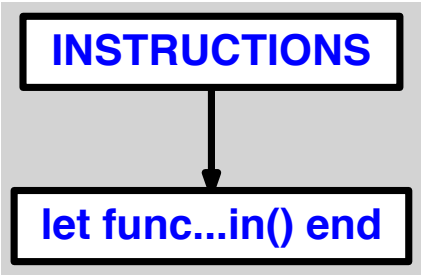
3.2.101 2 instructions sans parentheses

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


INSTRUCTIONS
let func...in() end

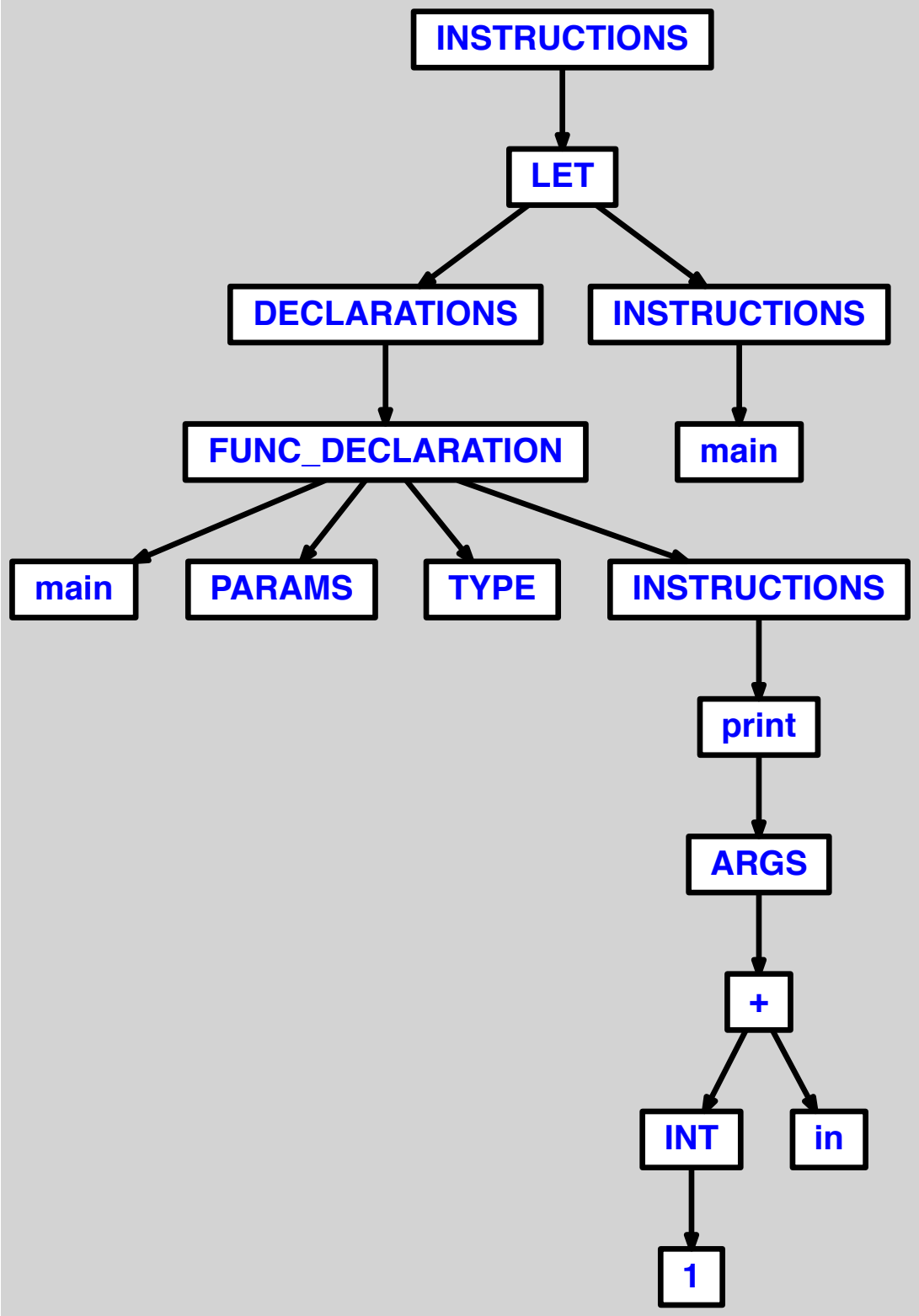
3.2.102 3 instructions sans parentheses

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


INSTRUCTIONS
let func...in() end

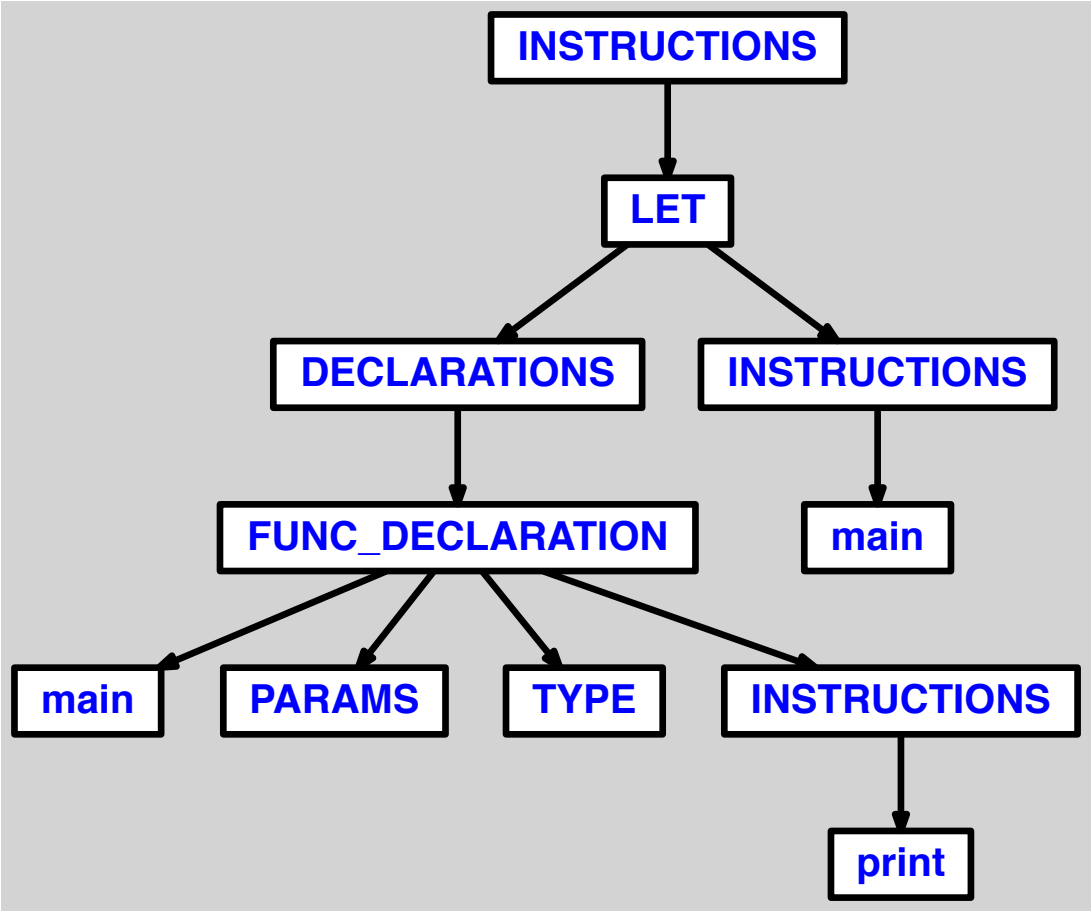
3.2.103 concatenation d'entier et de chaine

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


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

3.2.104 concatenation de 2 chaines

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

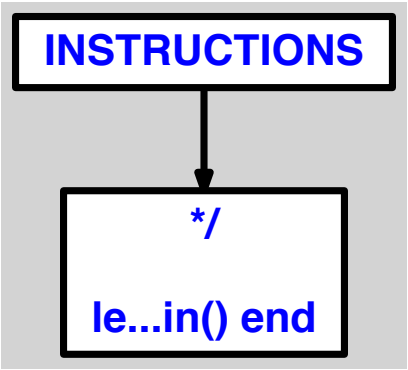

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
print

4 commentaire

4.1 KO

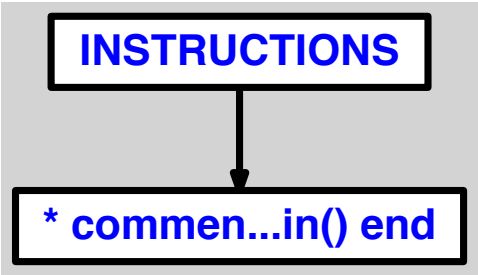
4.1.1 imbrication

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

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


INSTRUCTIONS
*/
le...in() end

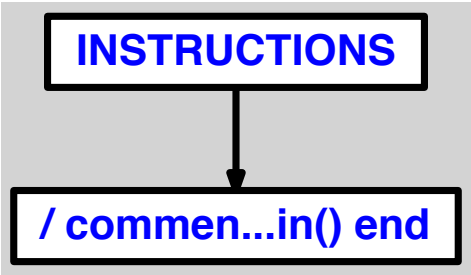
4.1.2 oubli de </> en debut de commentaire

```
* commentaire */

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


INSTRUCTIONS
* commen...in() end

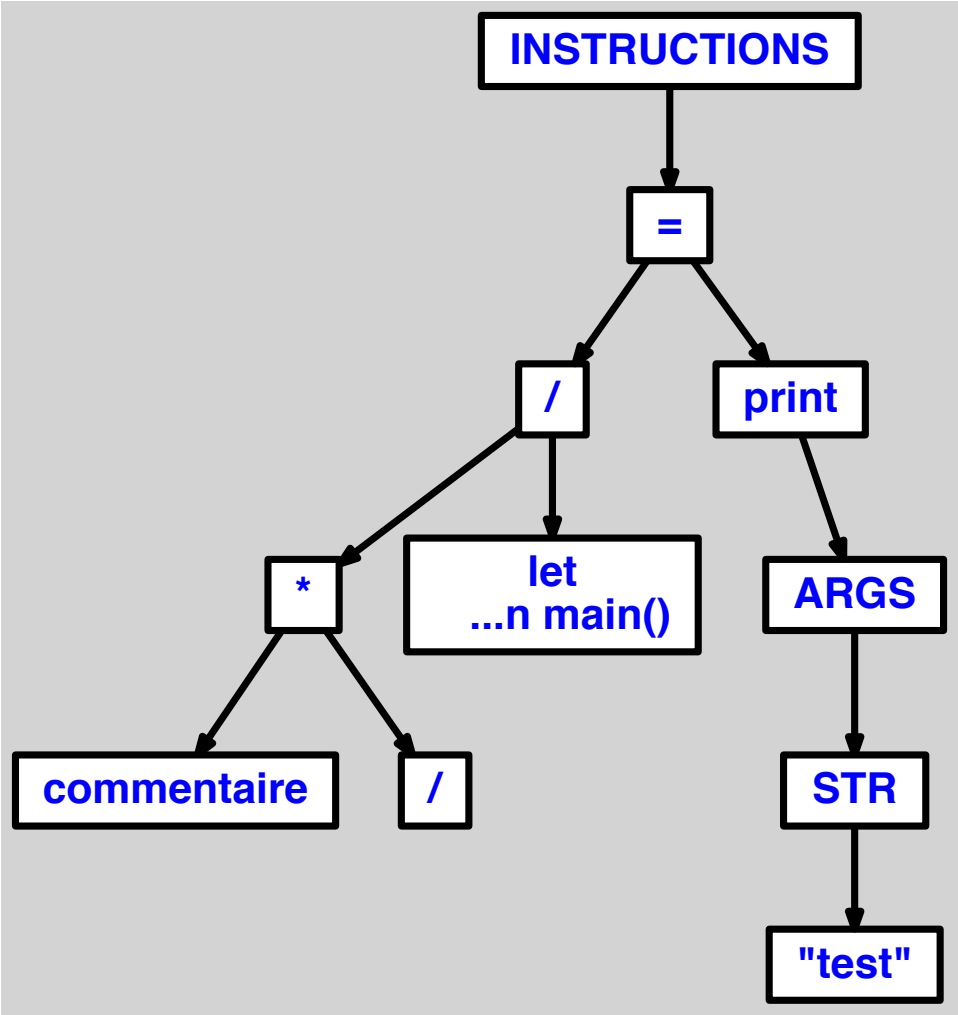
4.1.3 oubli de <*> en debut de commentaire

```
/ commentaire */

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


INSTRUCTIONS
/ commen...in() end

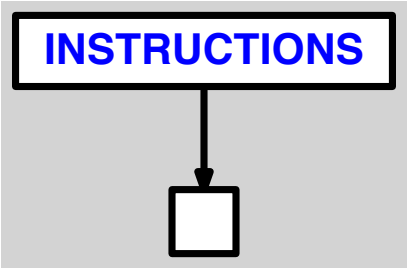
4.1.4 oubli de </*> en debut de commentaire

```
commentaire */

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


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

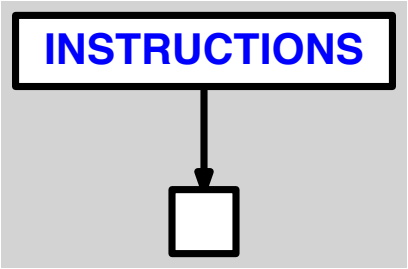
4.1.5 oubli de <*> en fin de commentaire

```
/* commentaire /

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


INSTRUCTIONS

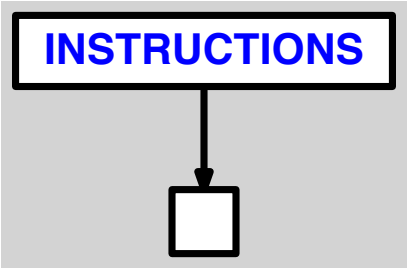
4.1.6 oubli de </> en fin de commentaire

```
/* commentaire *

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


INSTRUCTIONS

4.1.7 oubli de <*/> en fin de commentaire

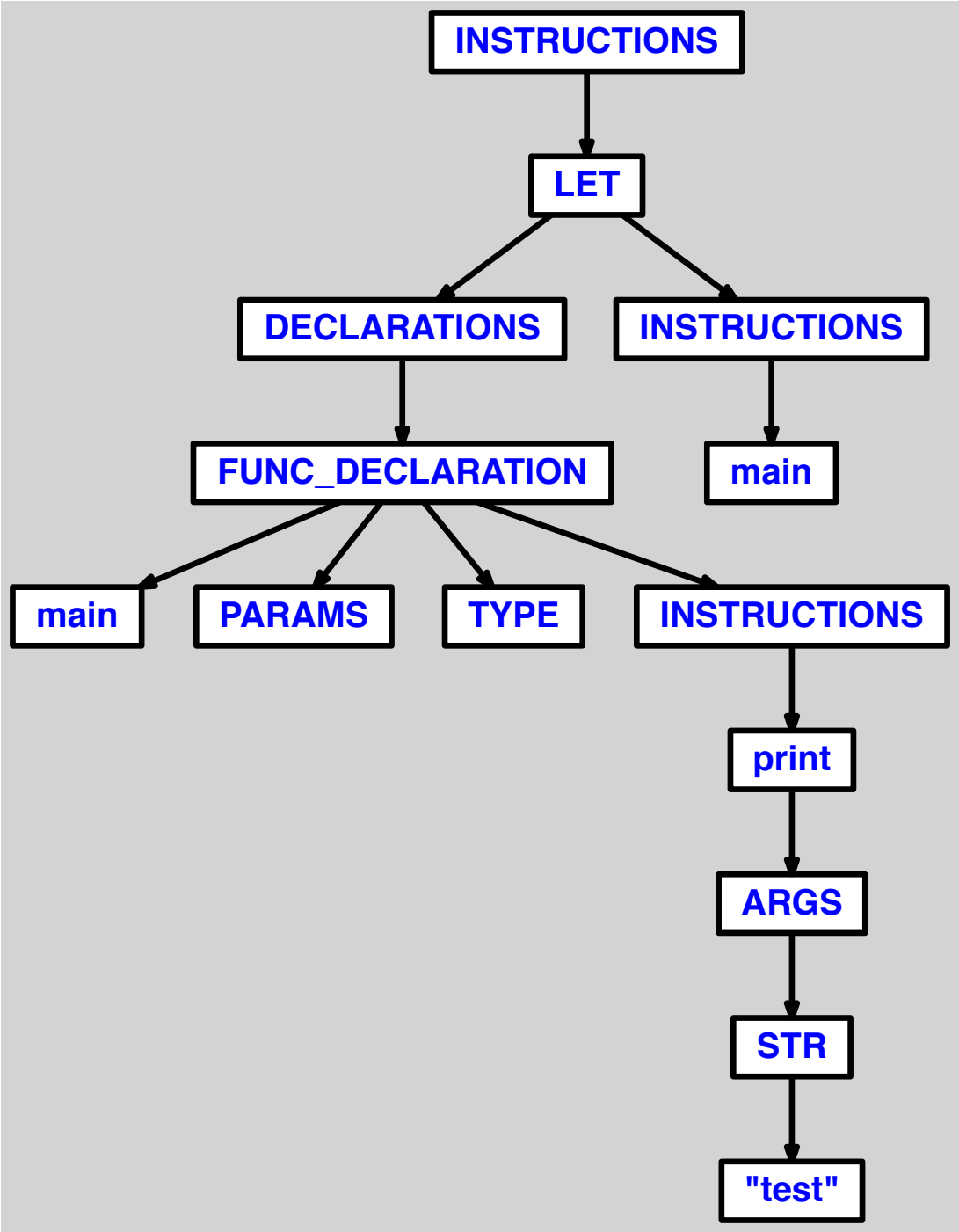
```
/* commentaire

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


INSTRUCTIONS

4.2 OK

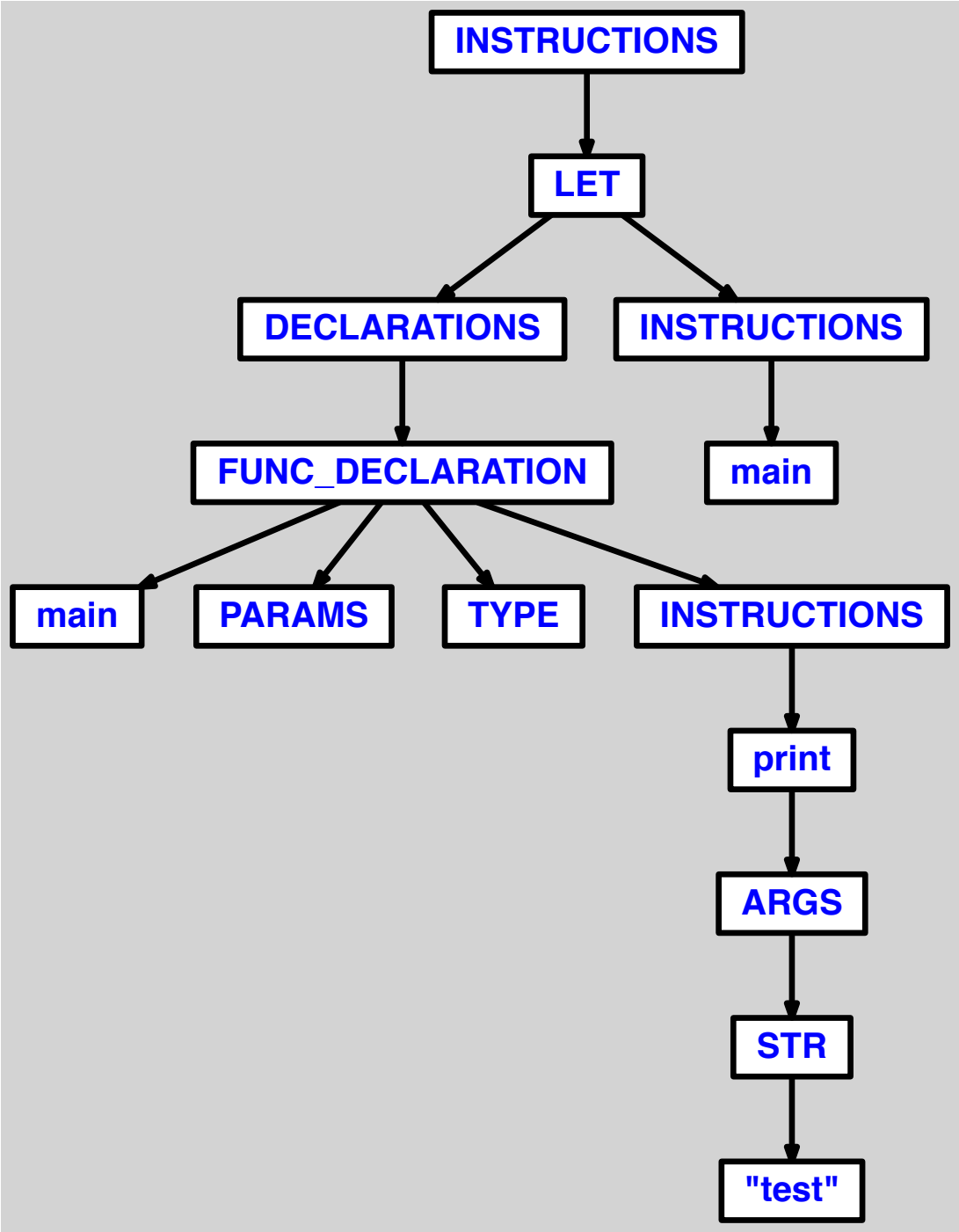
4.2.1 1 commentaire, sans instruction avant, sur 1 ligne

```
/* commentaire */

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


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

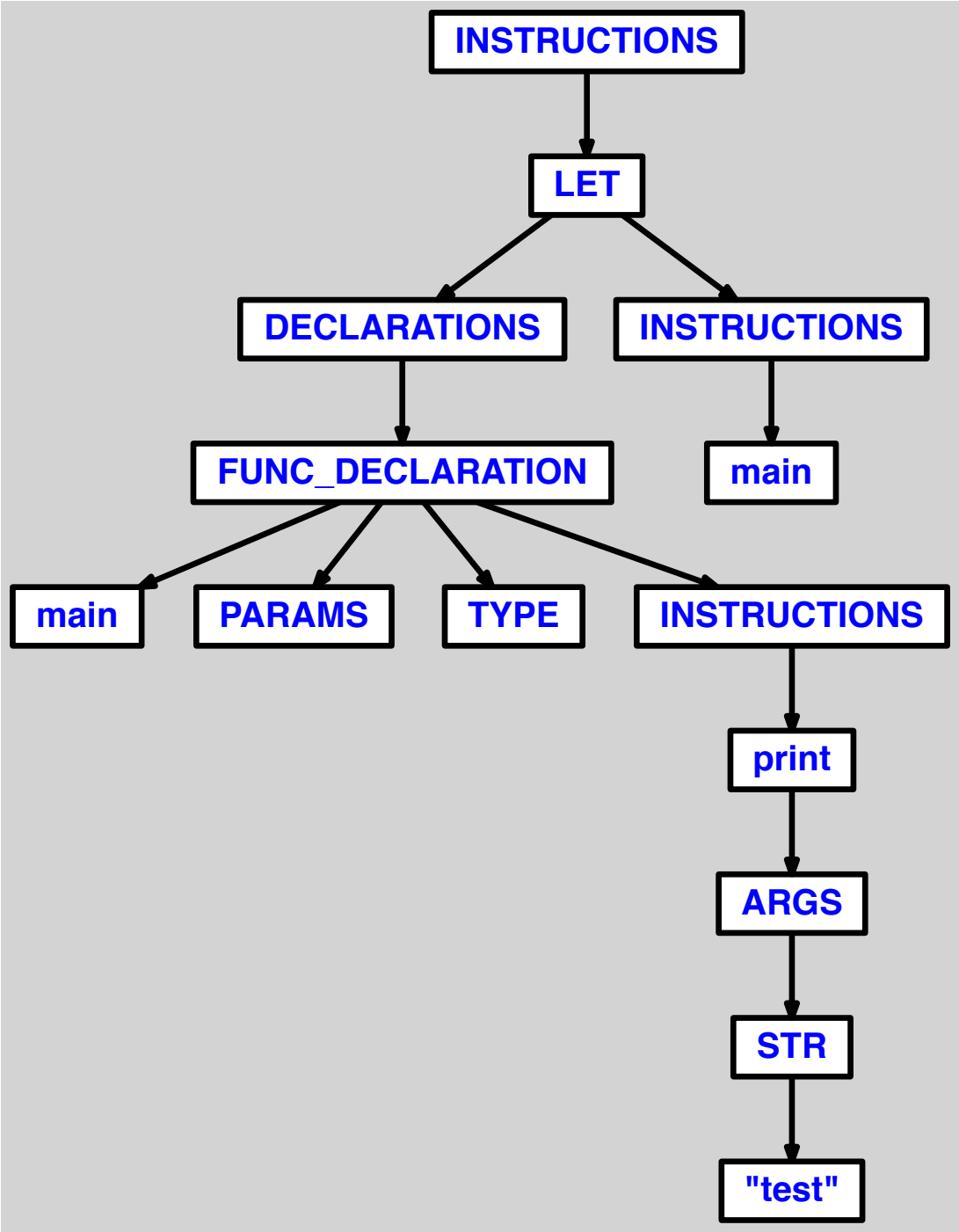
4.2.2 1 commentaire, sans instruction avant, sur plusieurs lignes

```
/*
1
2
3
*/

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


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

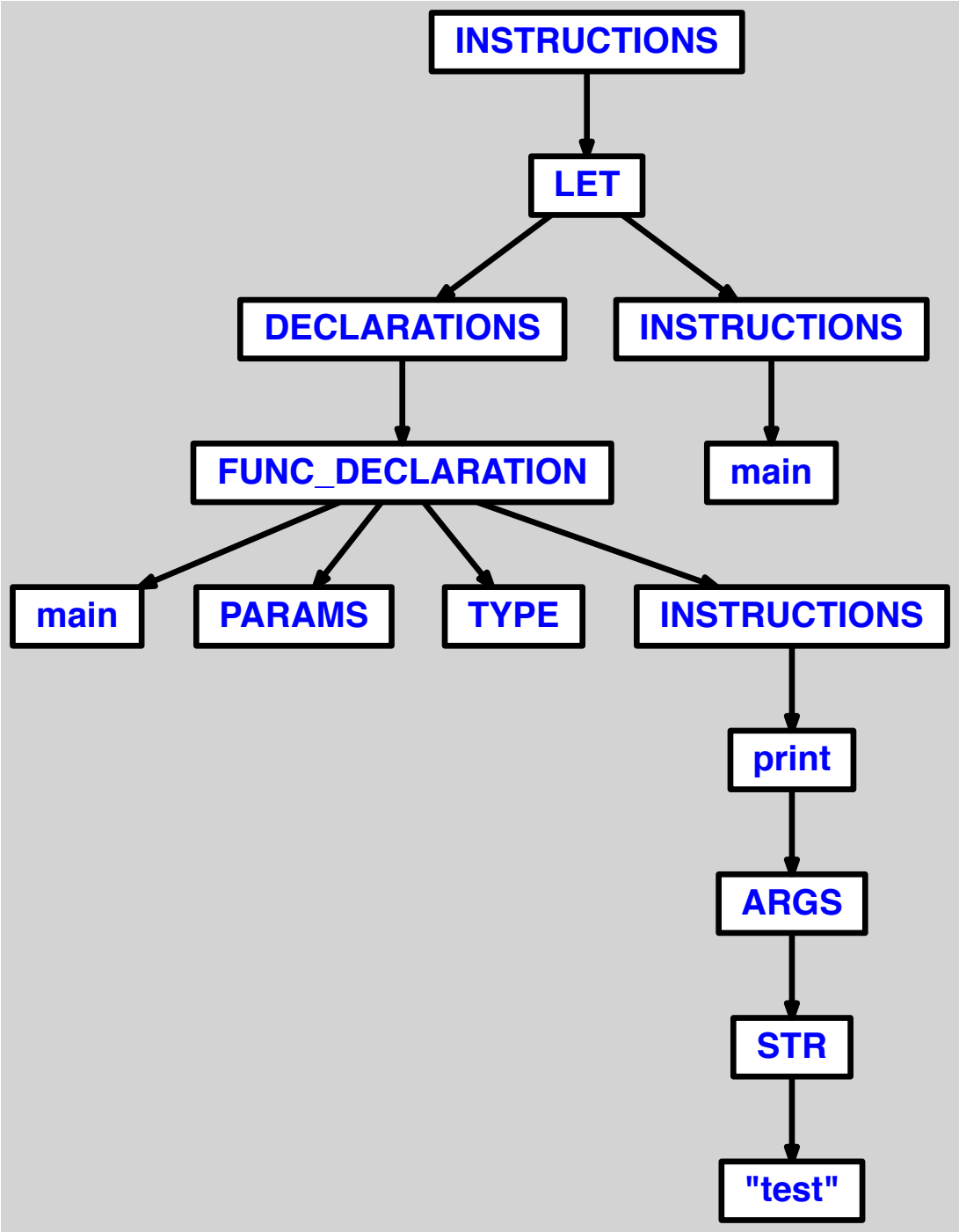
4.2.3 1 commentaire, apres instruction, sur 1 ligne

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


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

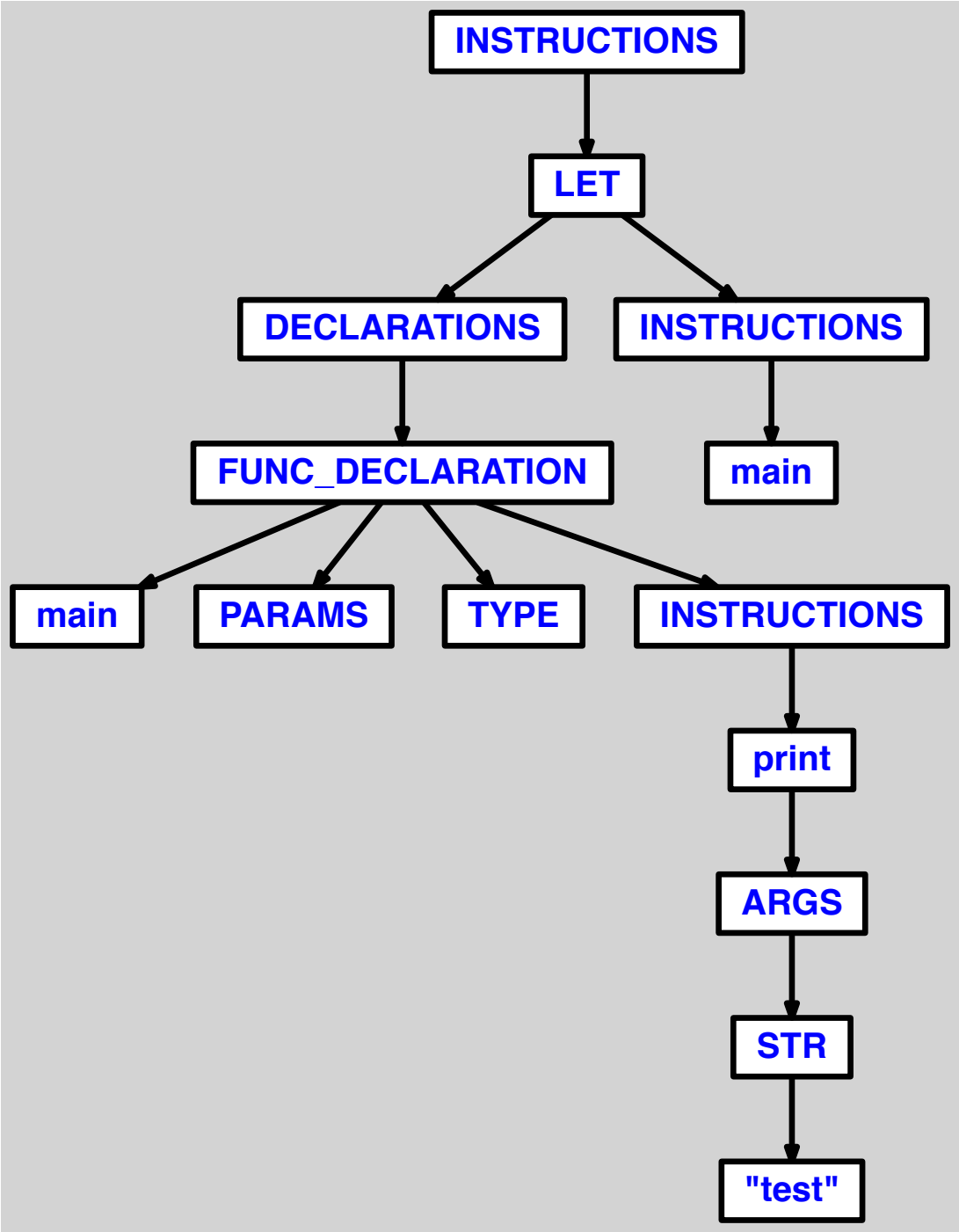
4.2.4 1 commentaire, apres instruction, sur plusieurs lignes

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


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

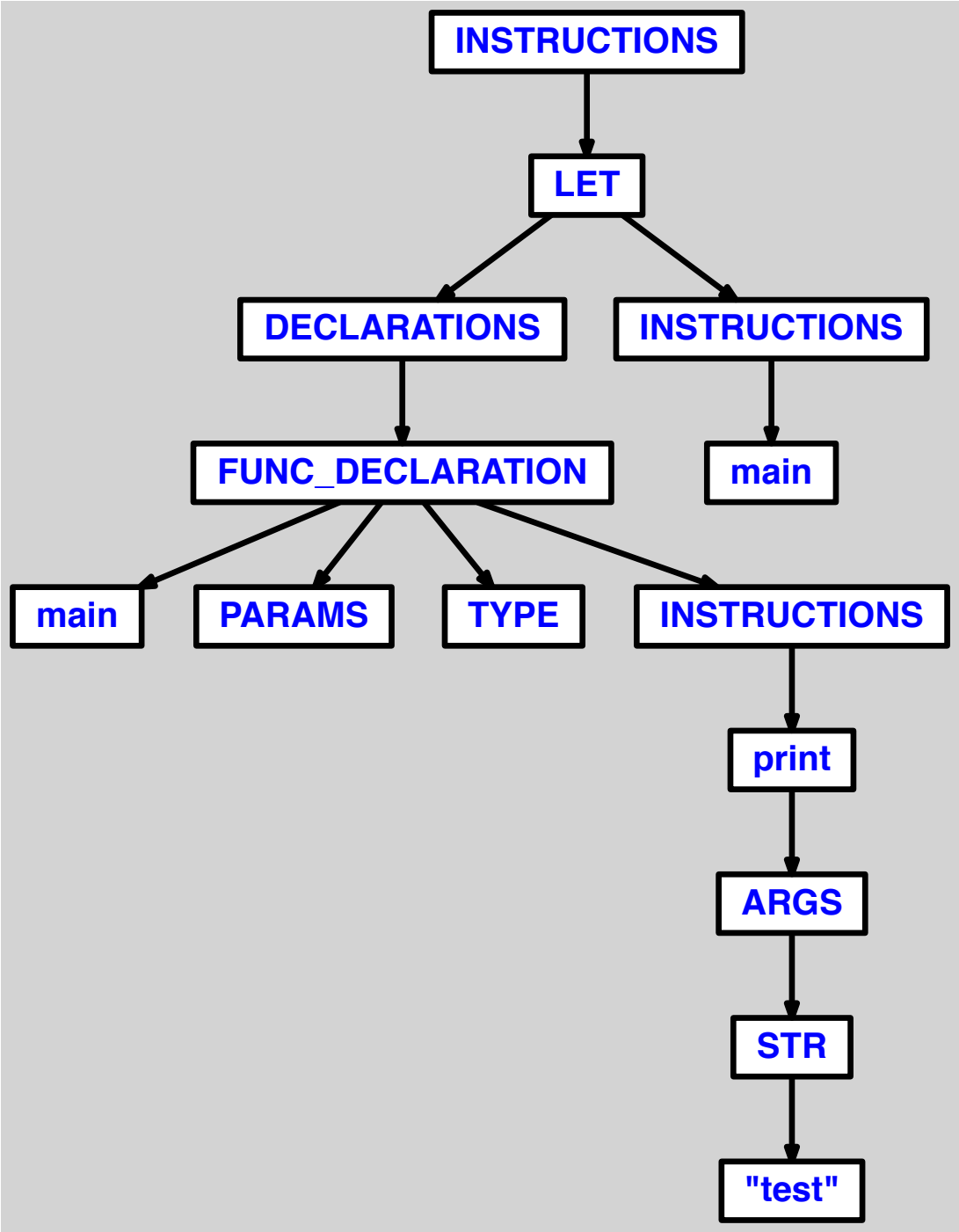
4.2.5 2 commentaires, sans instruction avant, sur 1 ligne

```
/* commentaire 1 */

/* commentaire 2 */

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


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

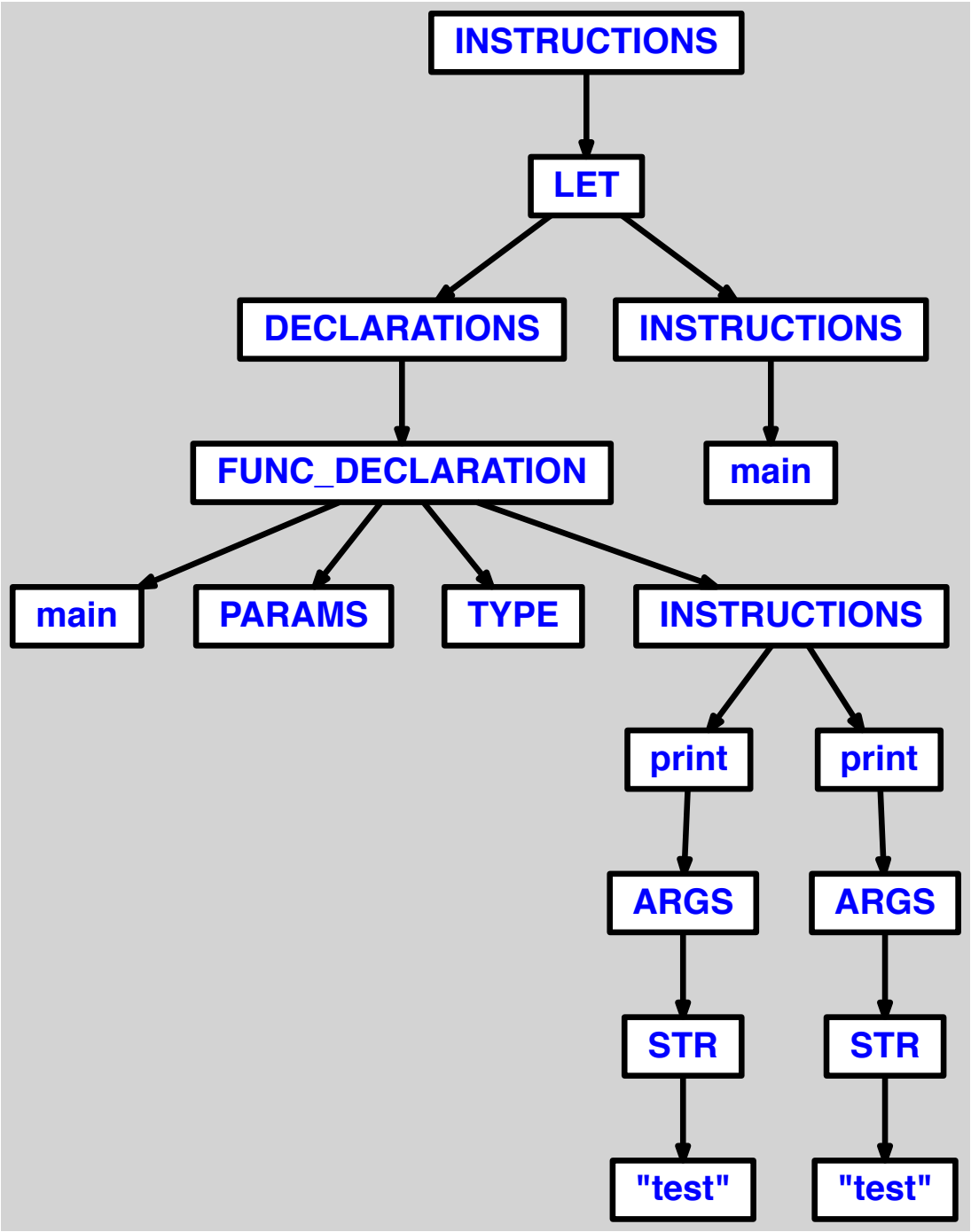
4.2.6 2 commentaires, sans instruction avant, sur plusieurs lignes

```
/*
1
2
3
*/

/*
4
5
6
*/

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


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

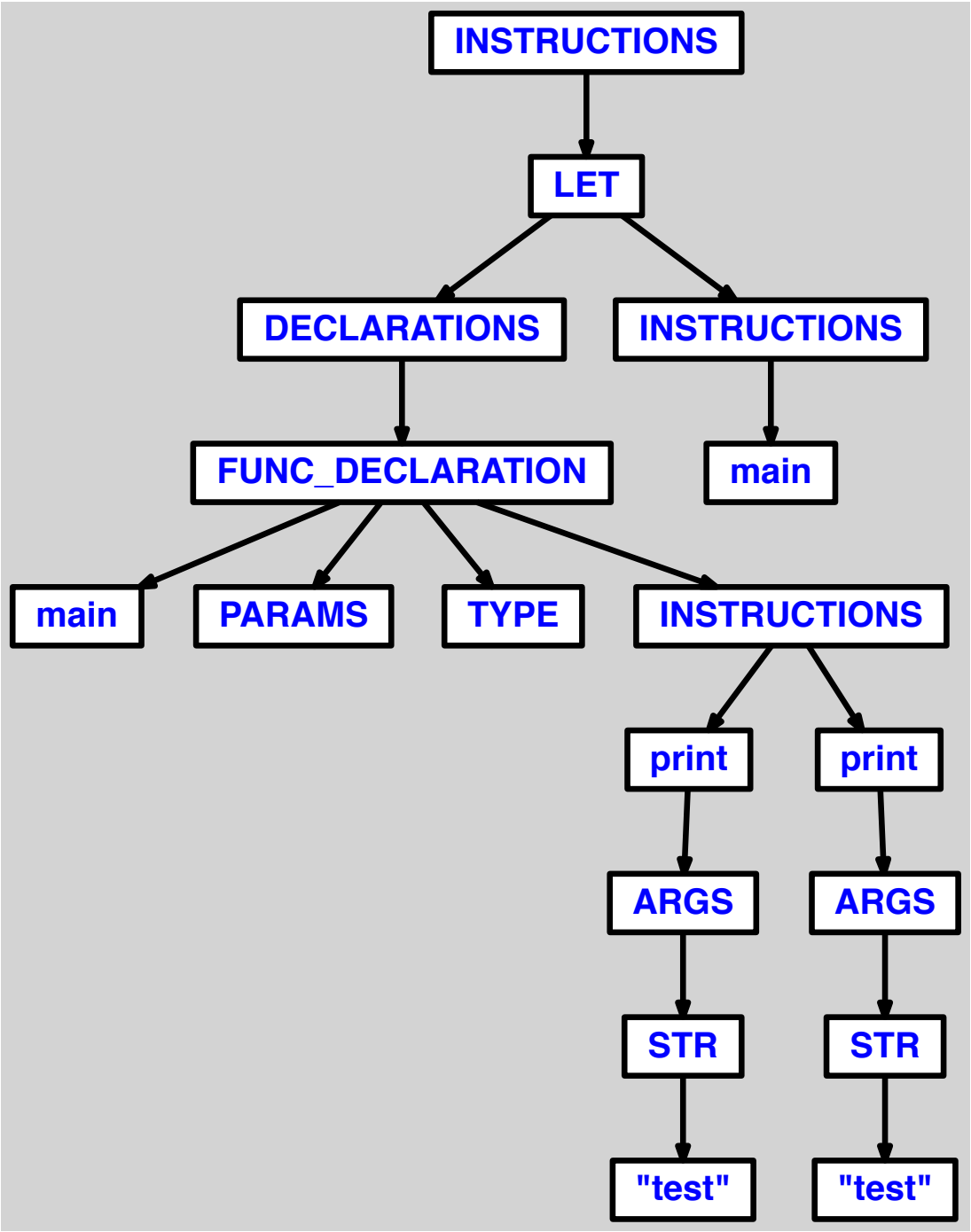
4.2.7 2 commentaires, apres instructions, sur 1 ligne

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


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

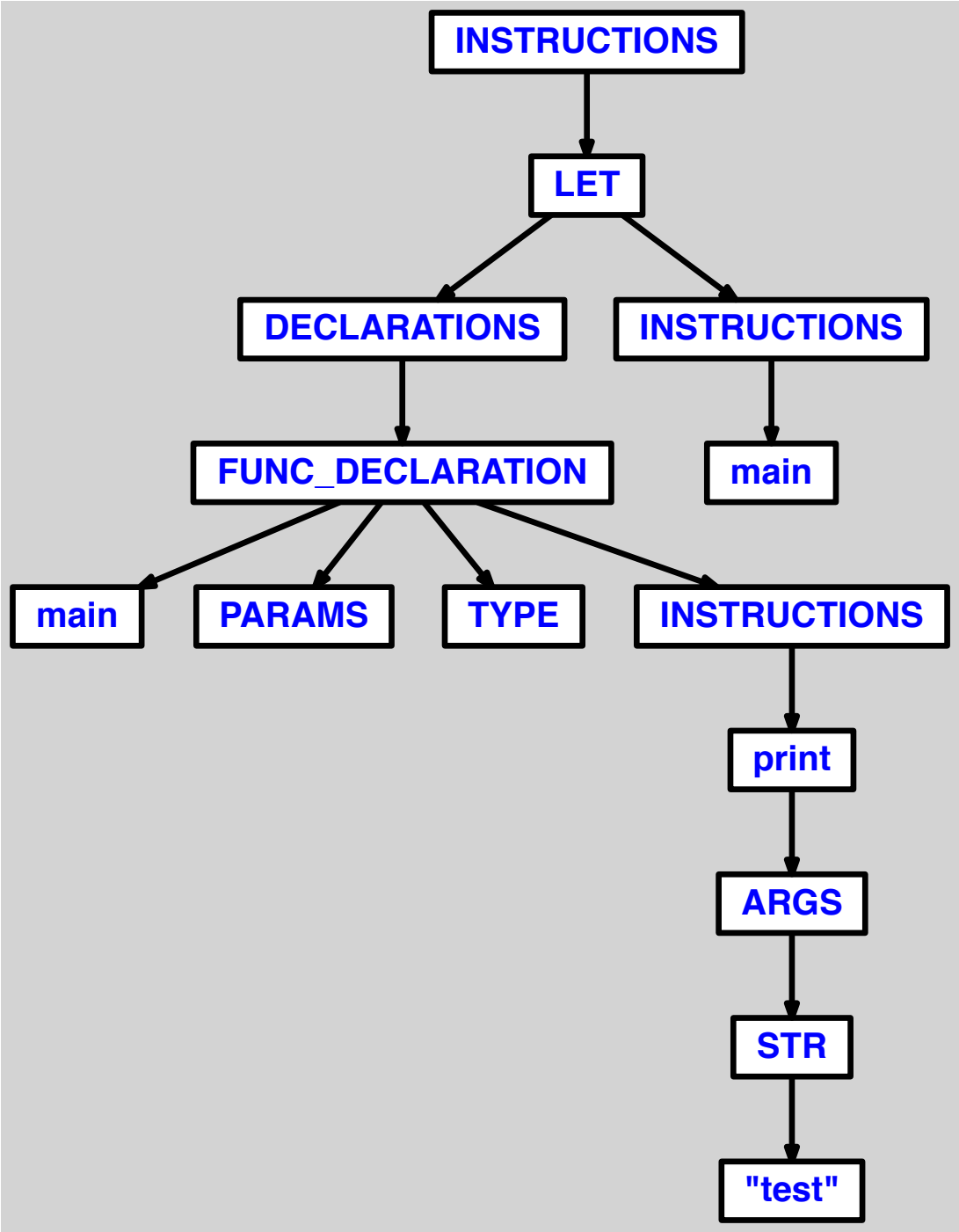
4.2.8 2 commentaires, apres instructions, sur plusieurs lignes

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


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

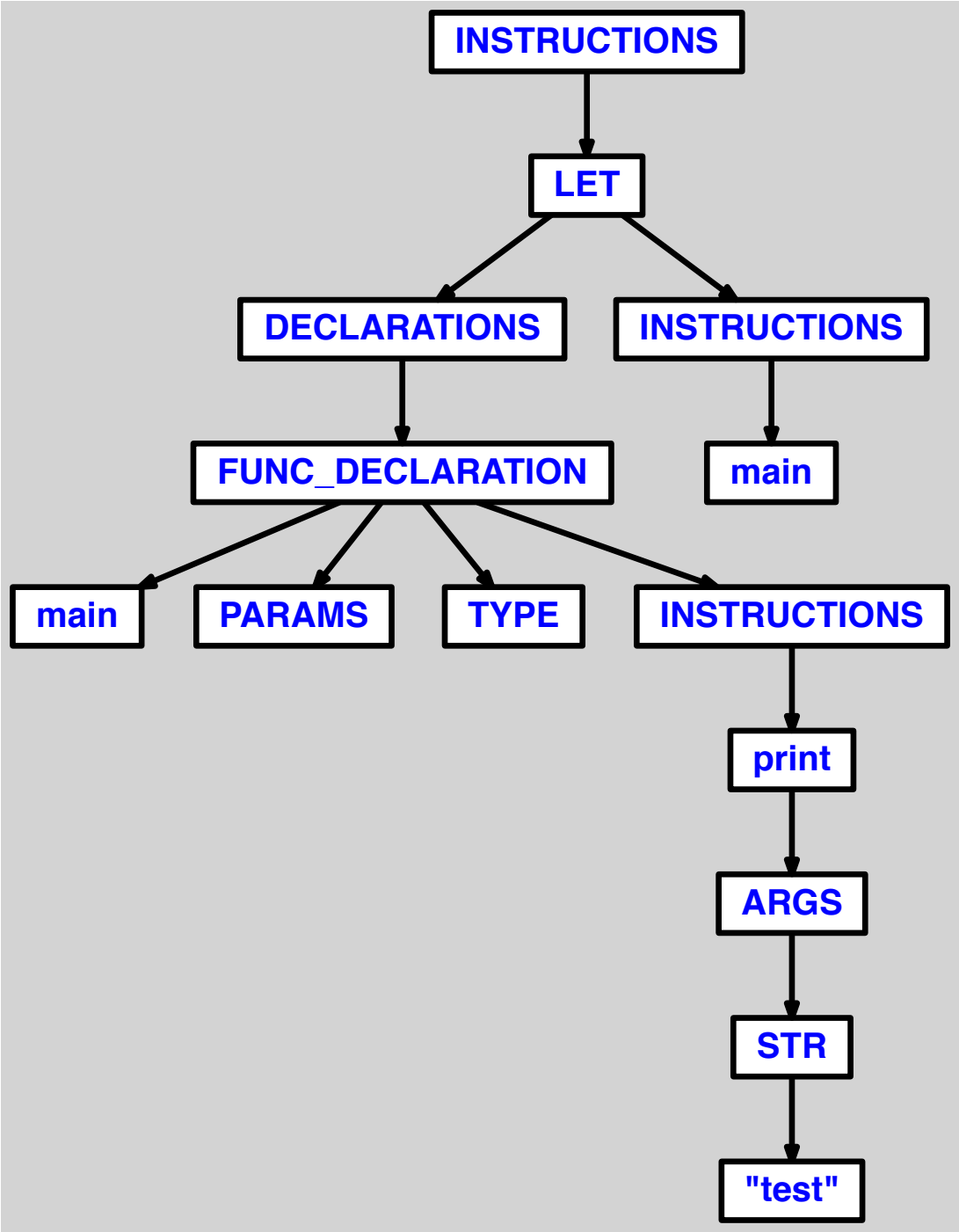
4.2.9 3 commentaires, sans instruction avant, sur 1 ligne

```
/* commentaire 1 */

/* commentaire 2 */

/* commentaire 3 */

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


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

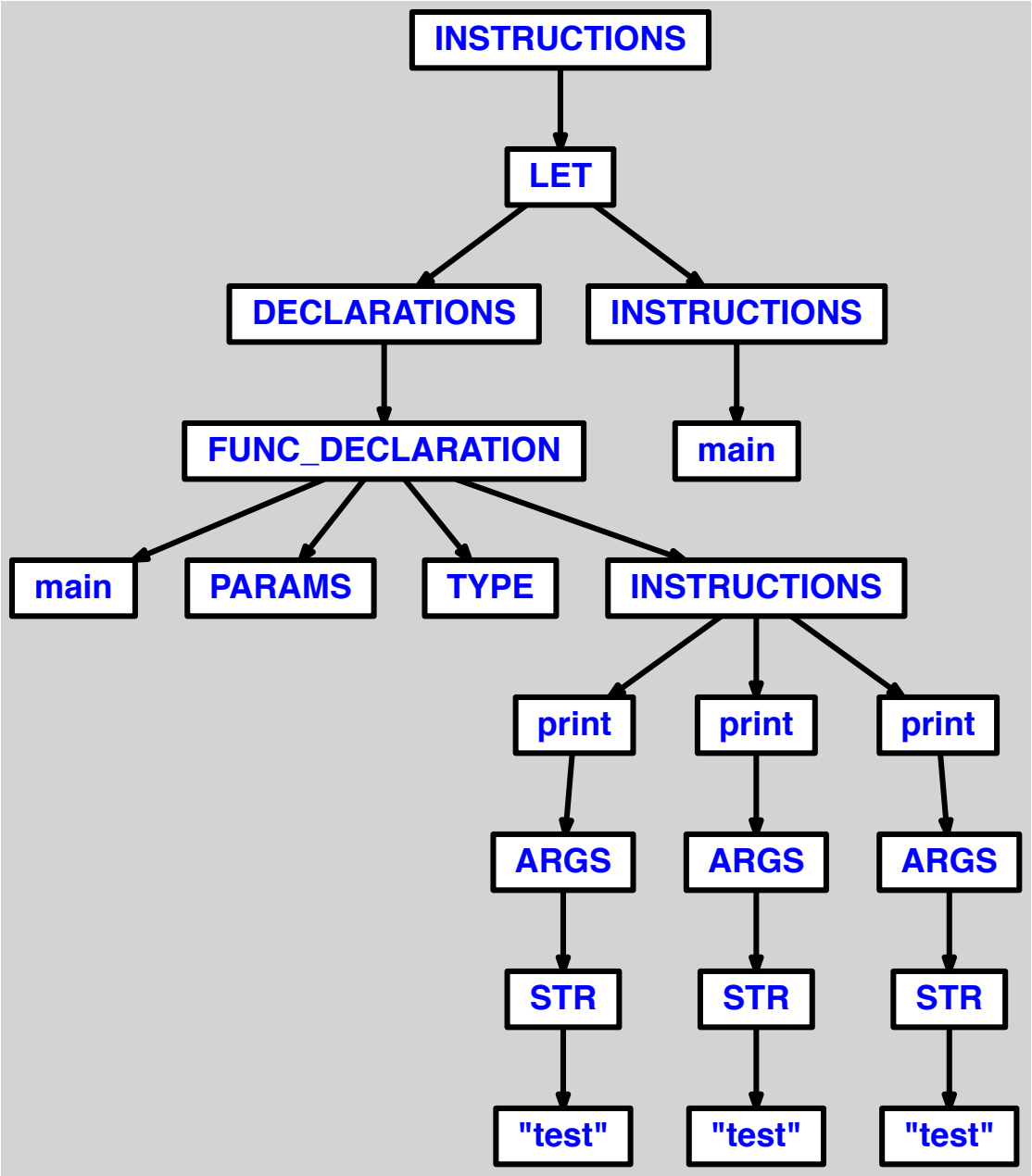
4.2.10 3 commentaires, sans instruction avant, sur plusieurs lignes

```
/*
1
2
3
*/

/*
4
5
6
*/

/*
7
8
9
*/

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


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

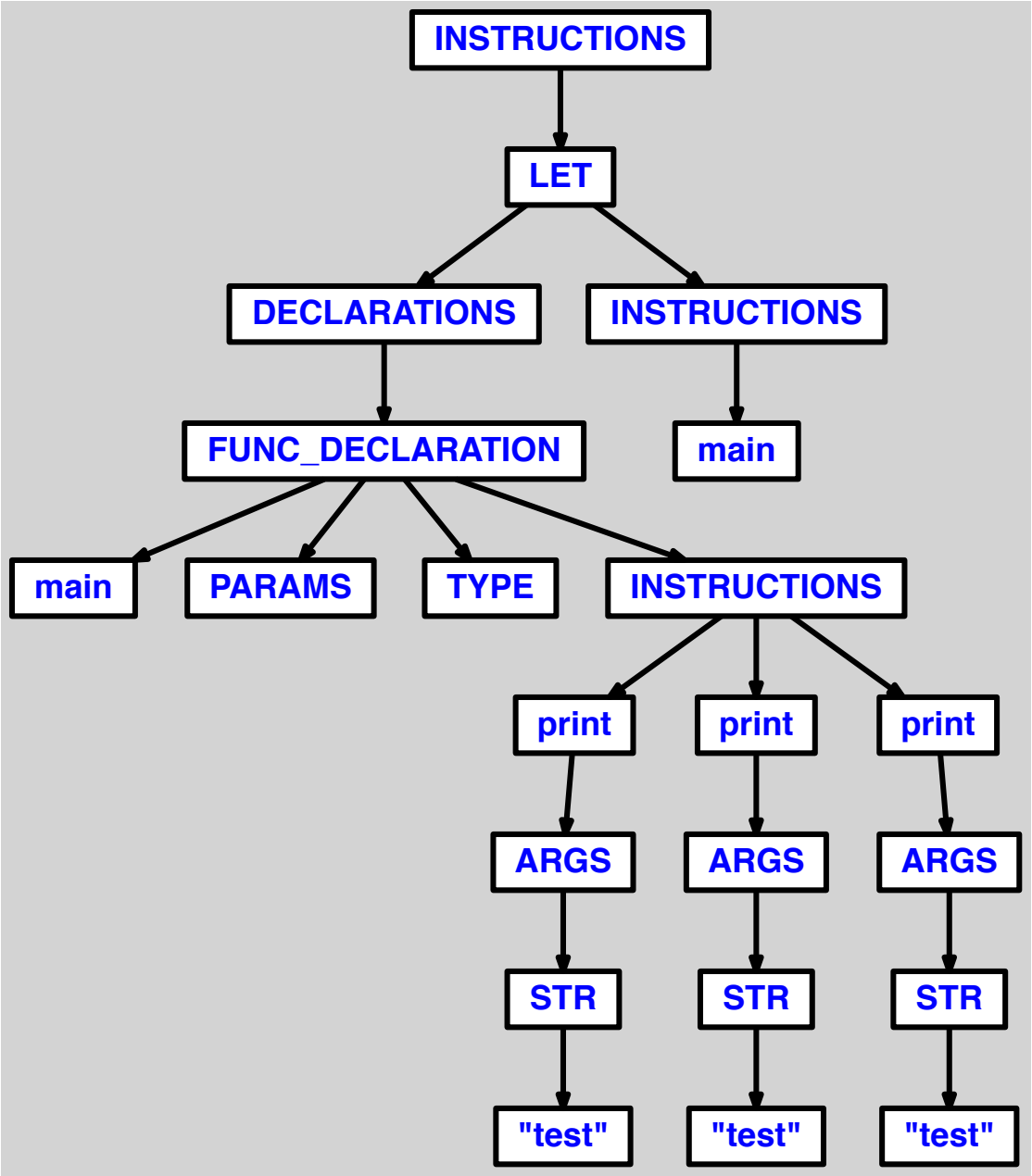
4.2.11 3 commentaires, apres instructions, sur 1 ligne

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


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

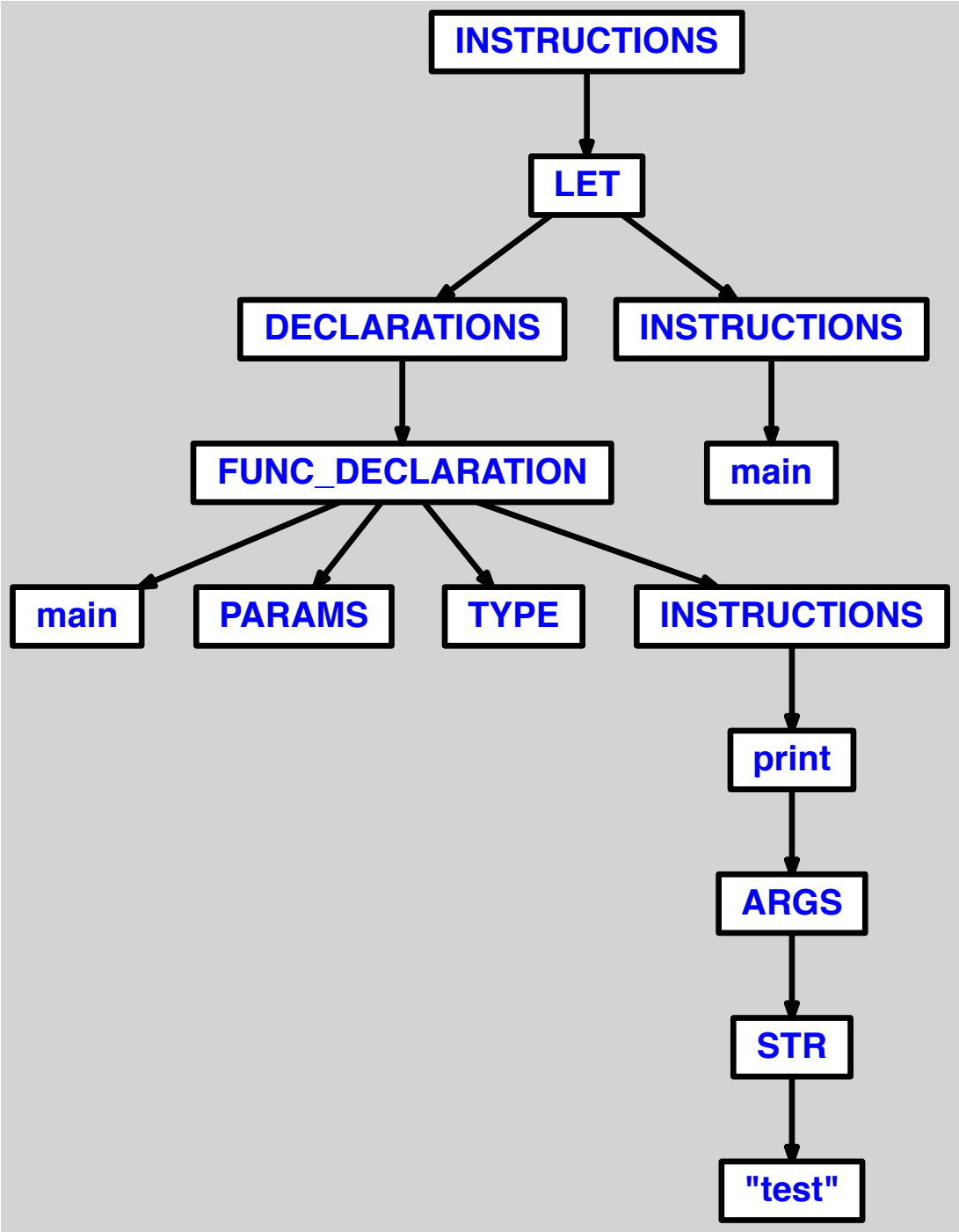
4.2.12 3 commentaires, apres instructions, sur plusieurs lignes

```
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
main
main
PARAMS
TYPE
INSTRUCTIONS
print
print
print
ARGS
ARGS
ARGS
STR
STR
STR
"test"
"test"
"test"

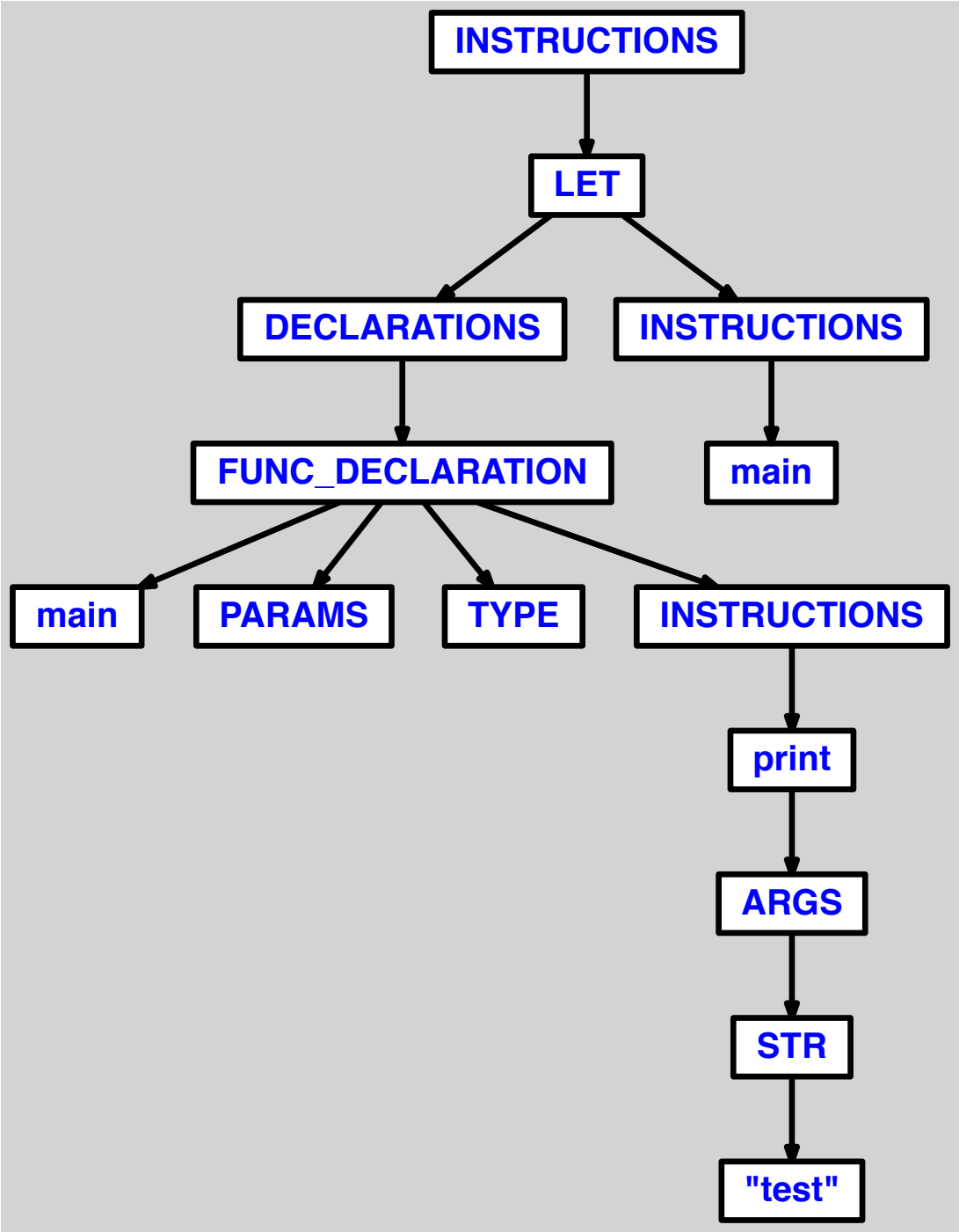
4.2.13 commentaire sans contenu sur 1 ligne

```
/* */

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


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

4.2.14 commentaire sans contenu sur plusieurs lignes

```
/*



*/

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

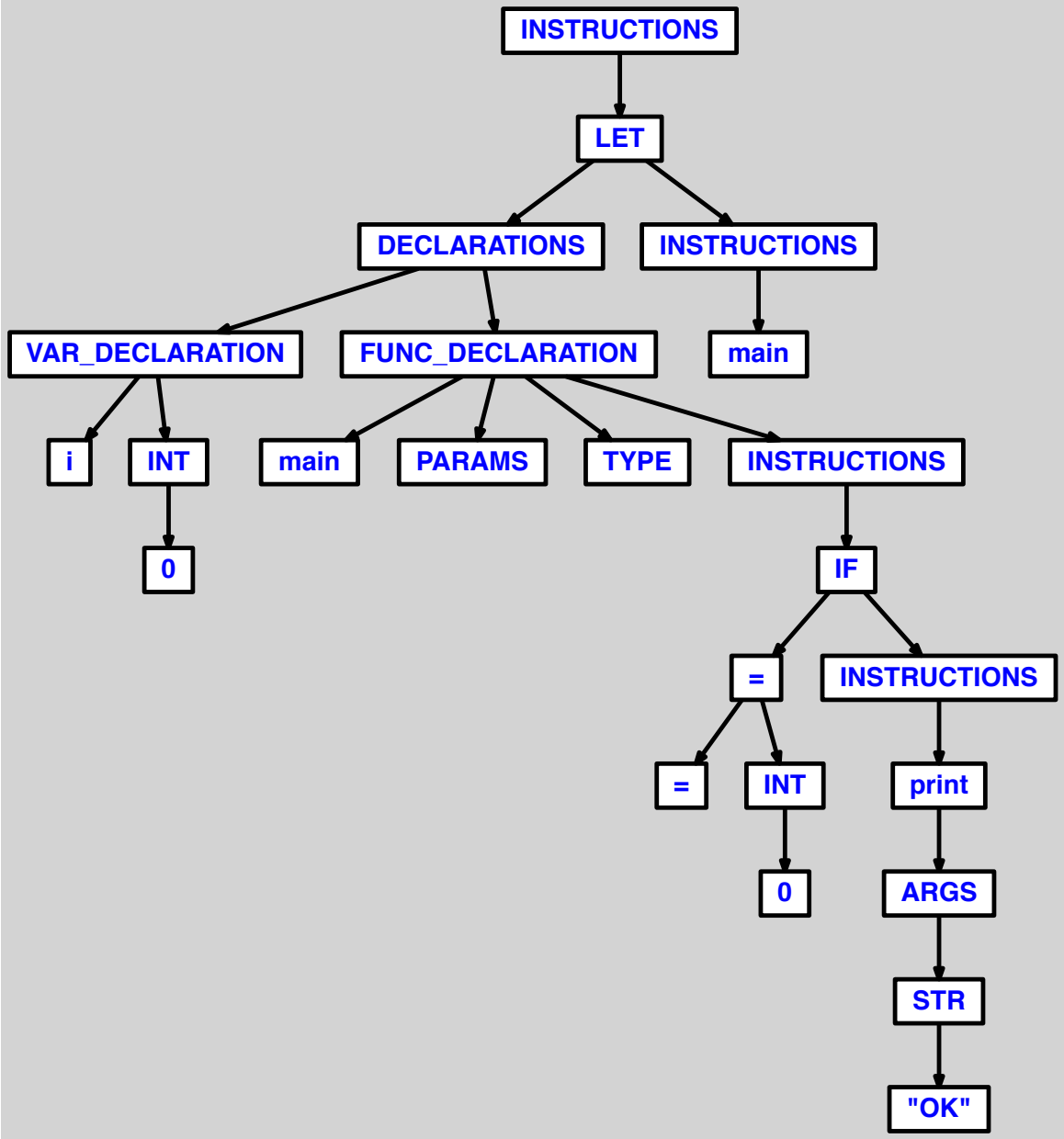

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
print
ARGS
STR
"test"

5 comparaison

5.1 KO

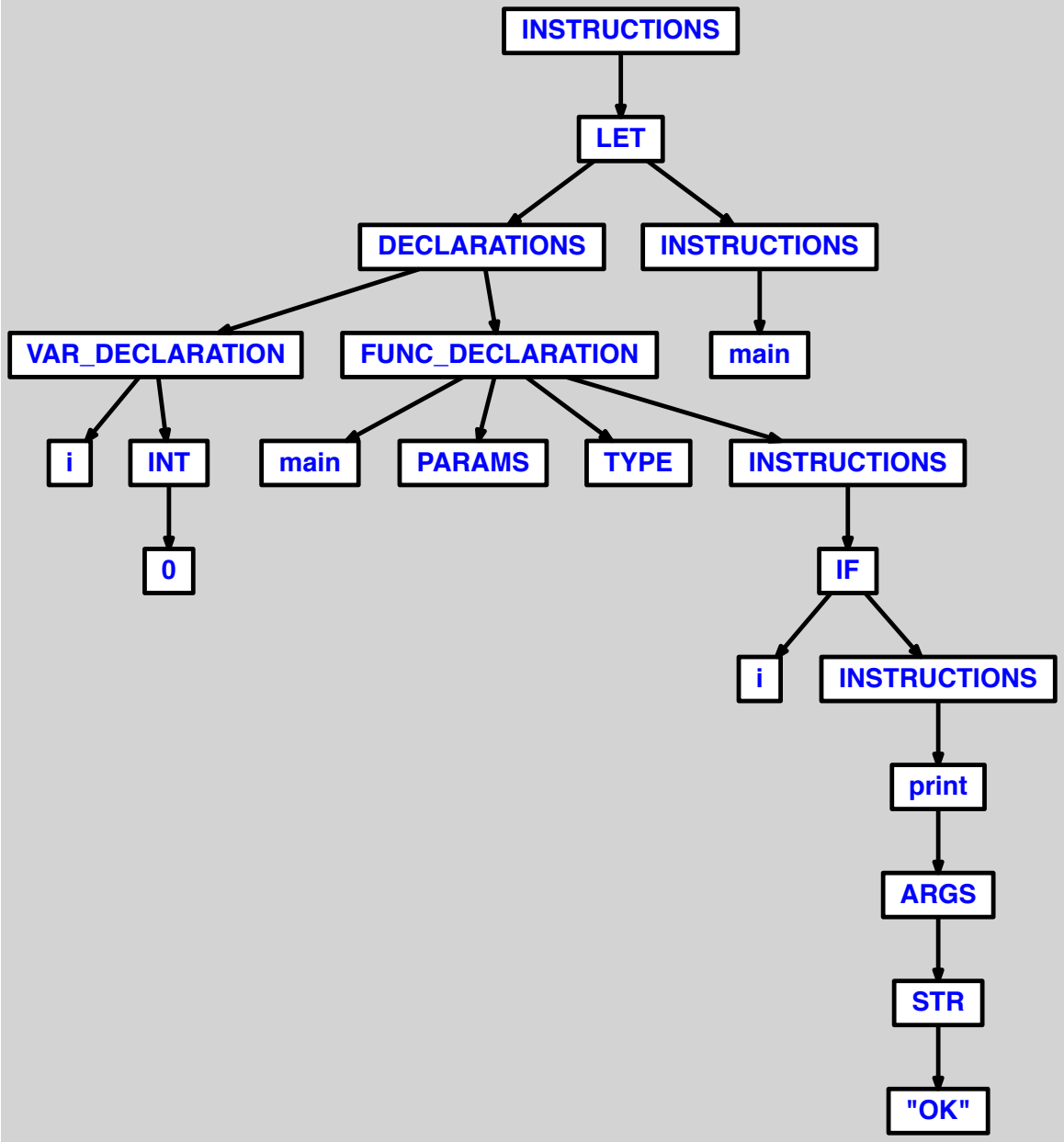
5.1.1 comparaison avec oubli d'operande gauche

```
let
  var i := 0

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


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

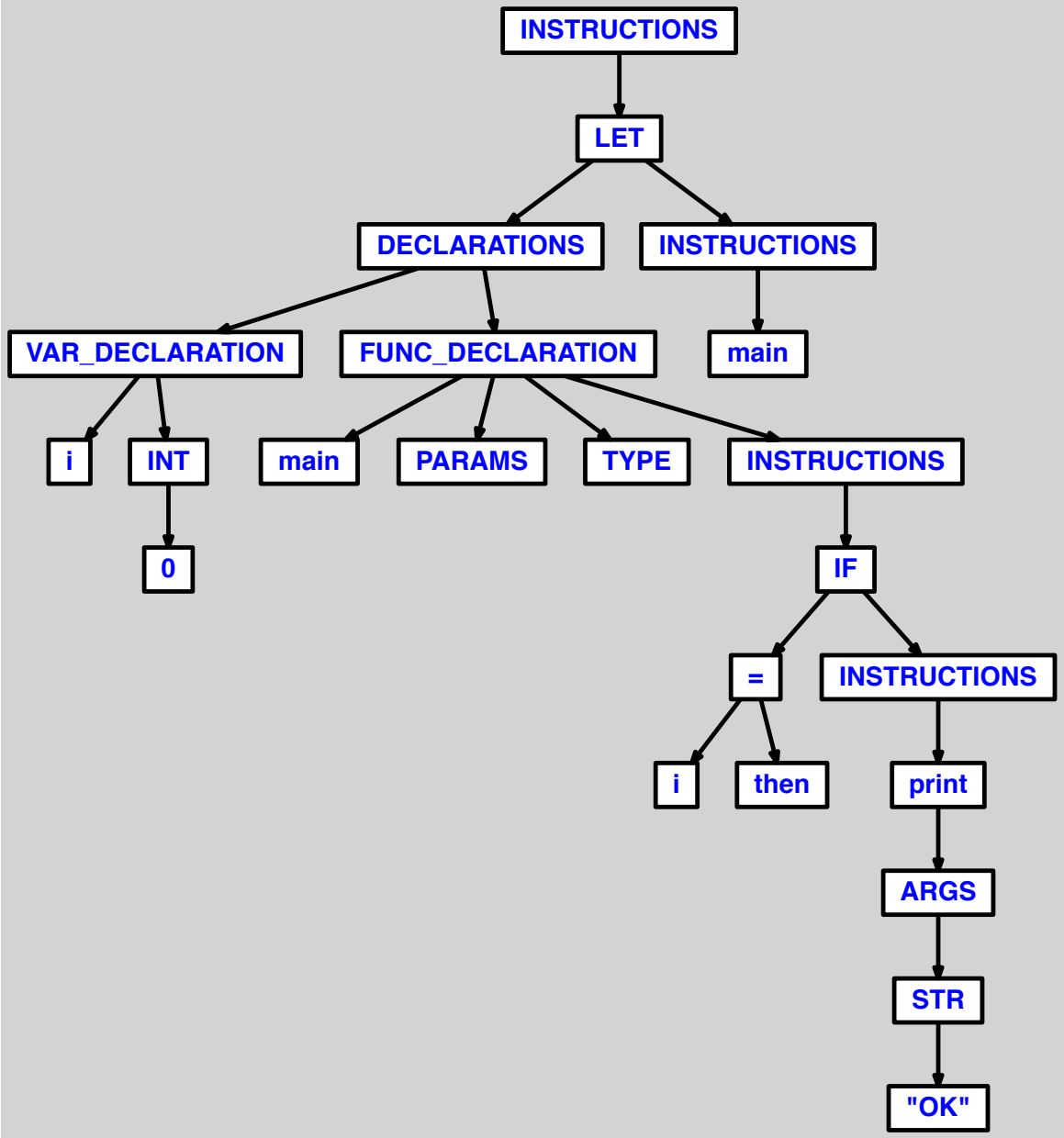
5.1.2 comparaison avec oubli d'operateur

```
let
  var i := 0

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


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

5.1.3 comparaison avec oubli d'operateur droit

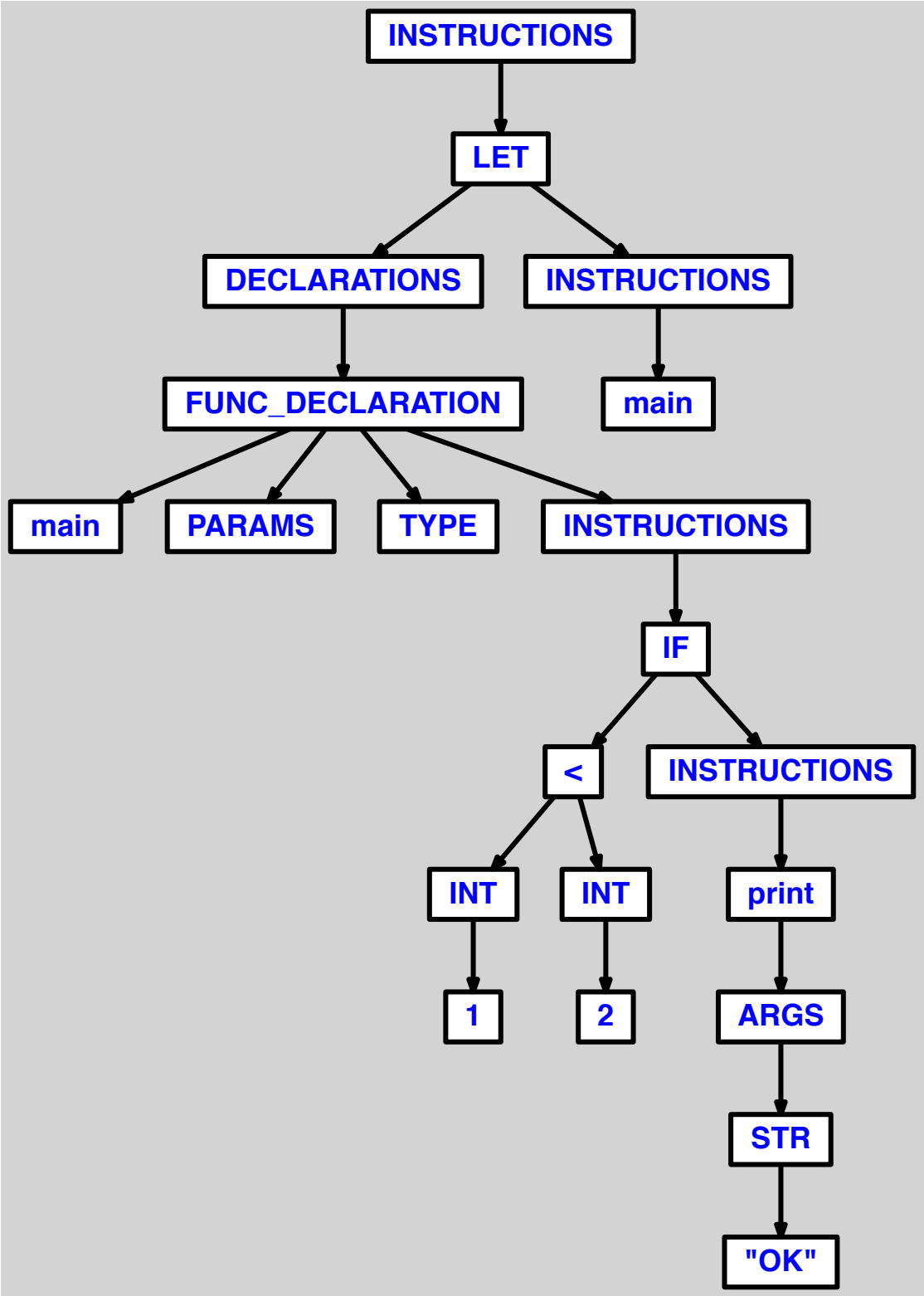
```
let
  var i := 0

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


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

5.2 OK

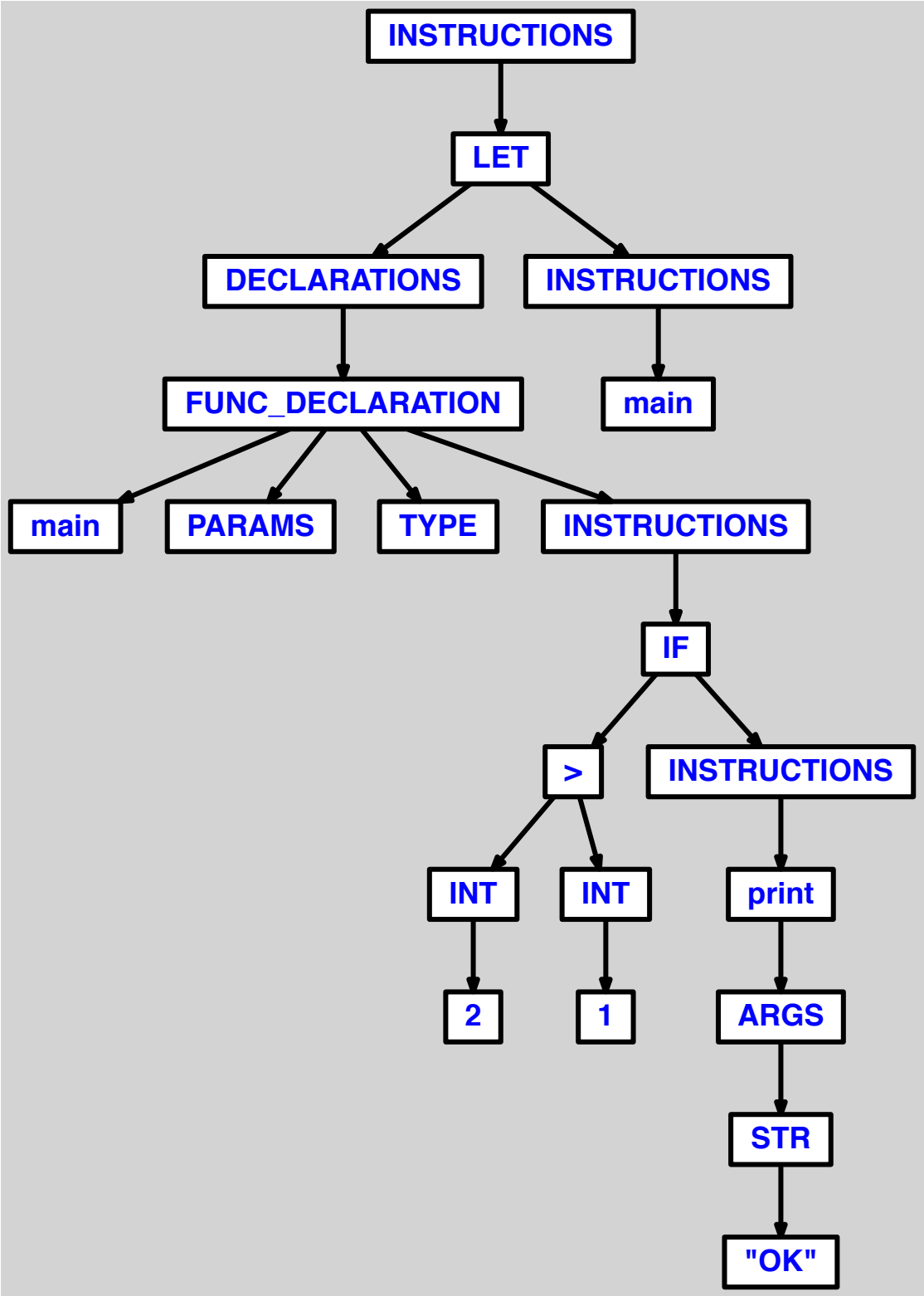
5.2.1 comparaison simple d'entiers avec <"<">

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


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

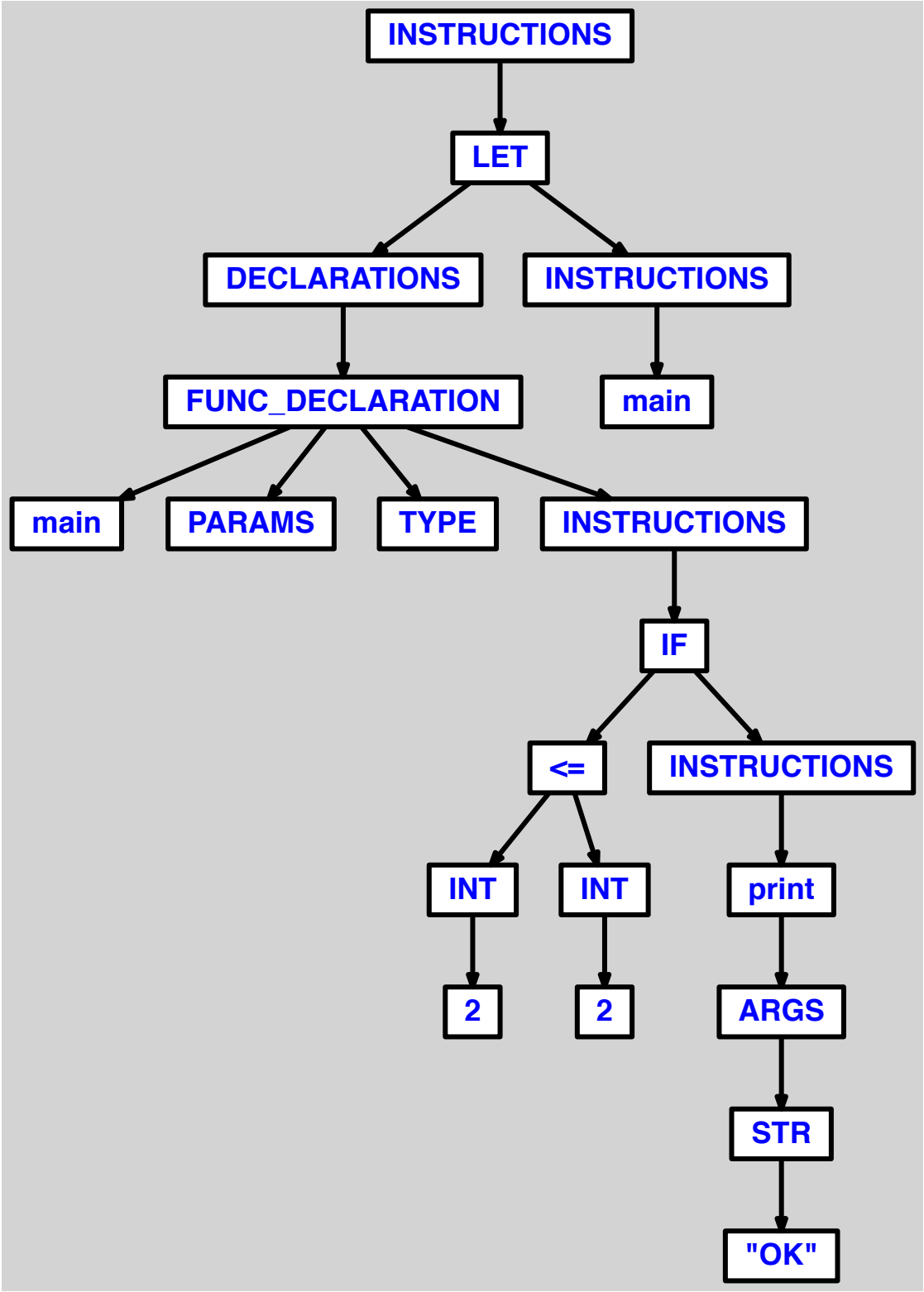
5.2.2 comparaison simple d'entiers avec <">">

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


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

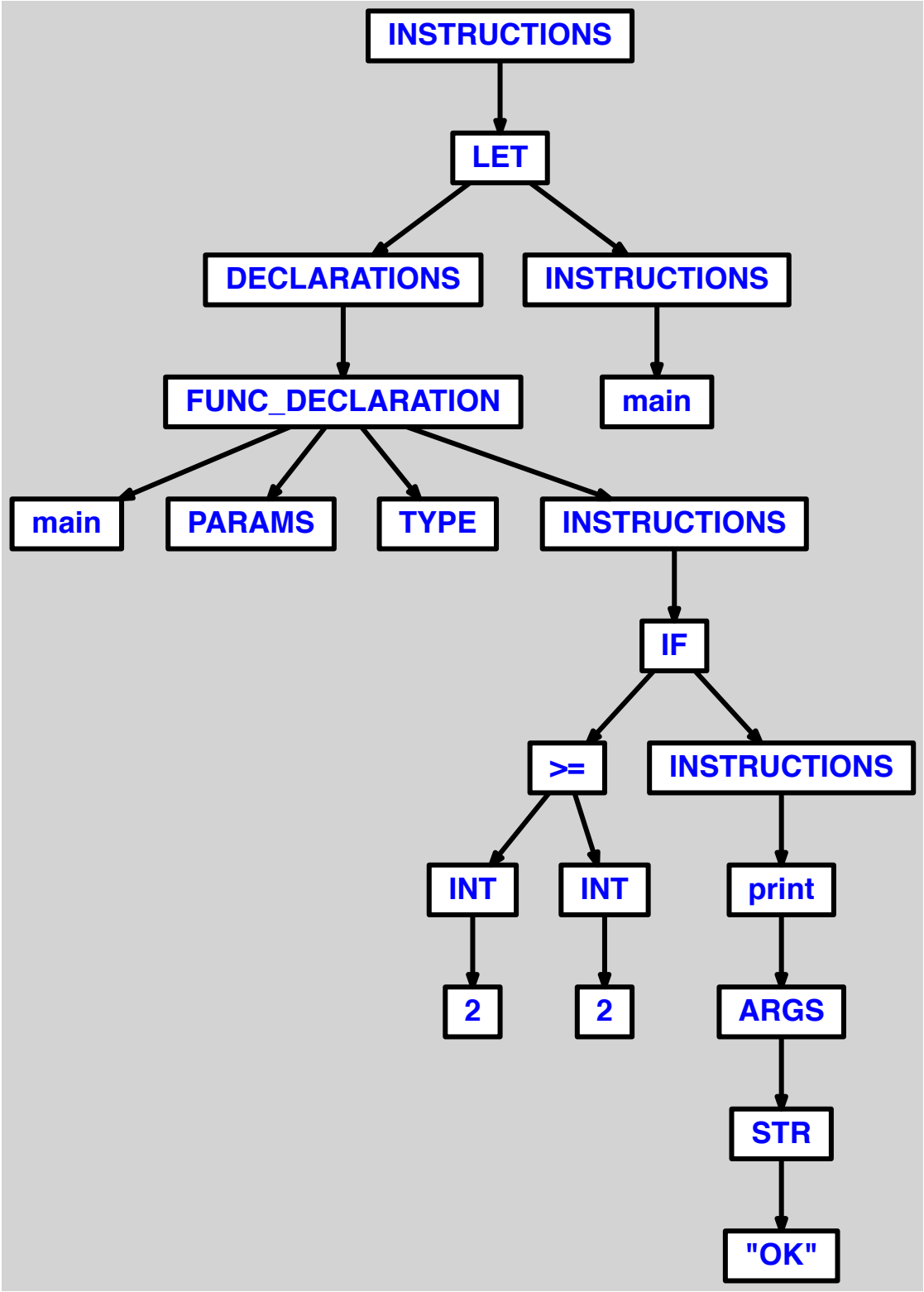
5.2.3 comparaison simple d'entiers avec <"<="><

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


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

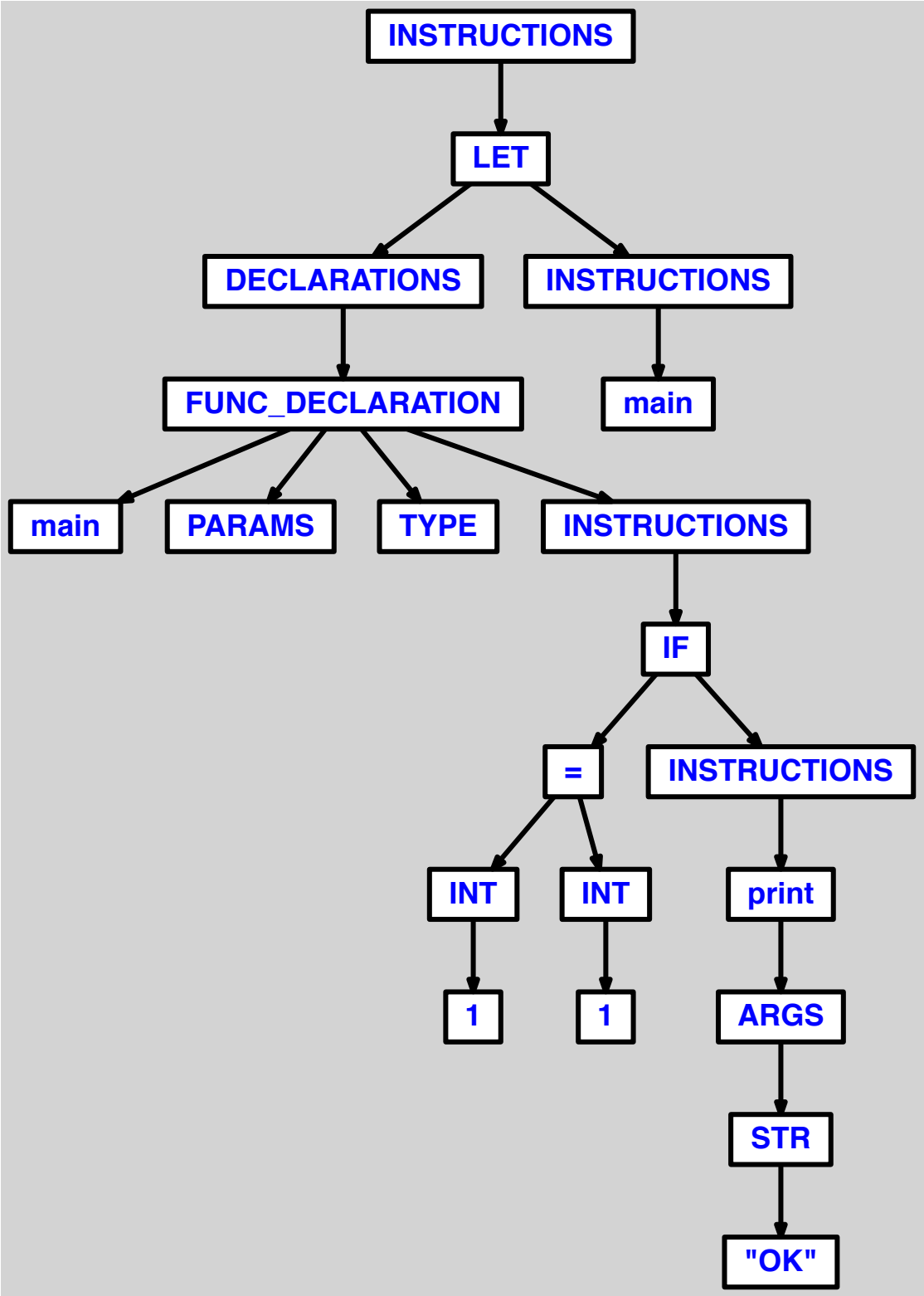
5.2.4 comparaison simple d'entiers avec <">="><

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


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

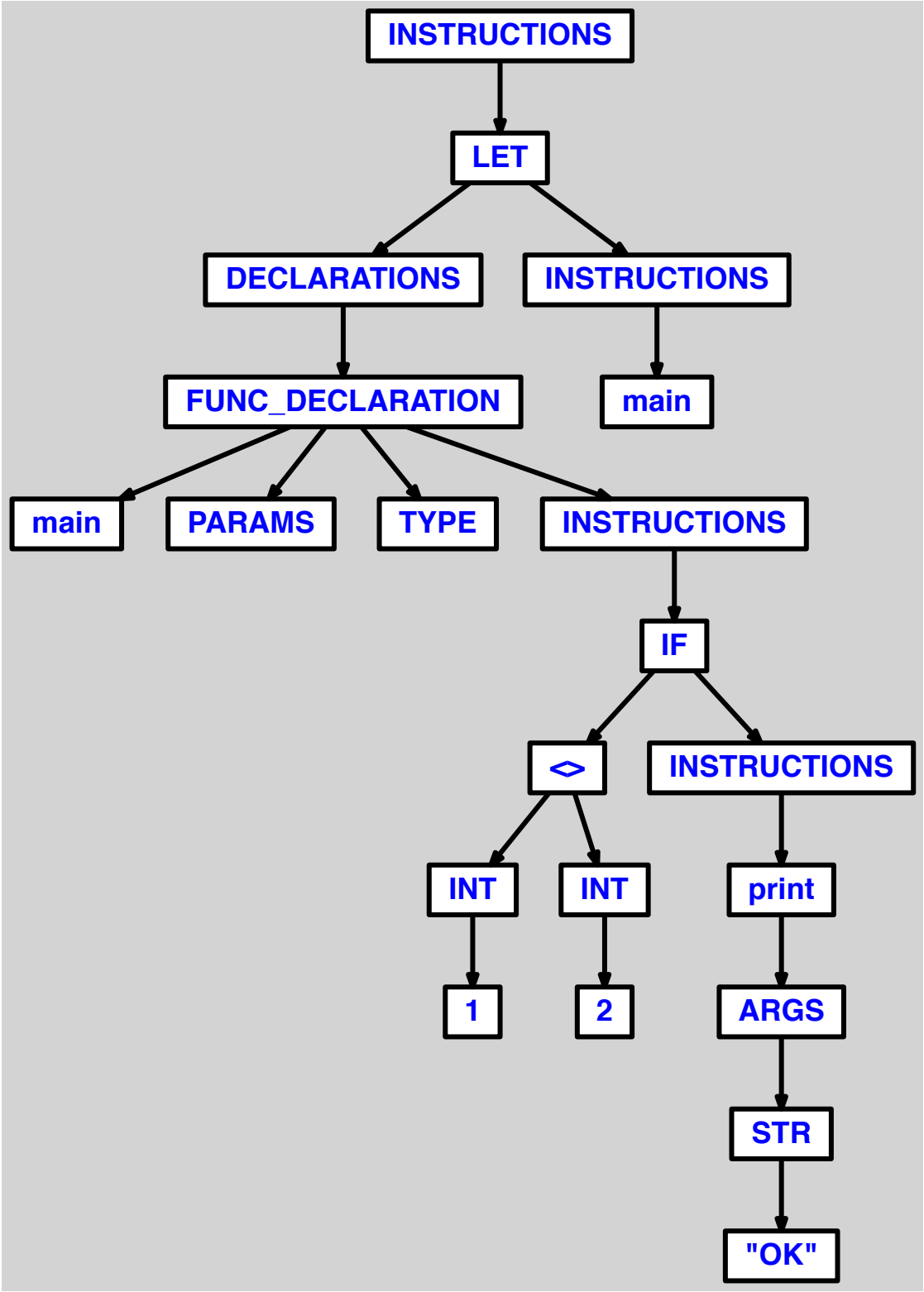
5.2.5 comparaison simple d'entiers avec <=>

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


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

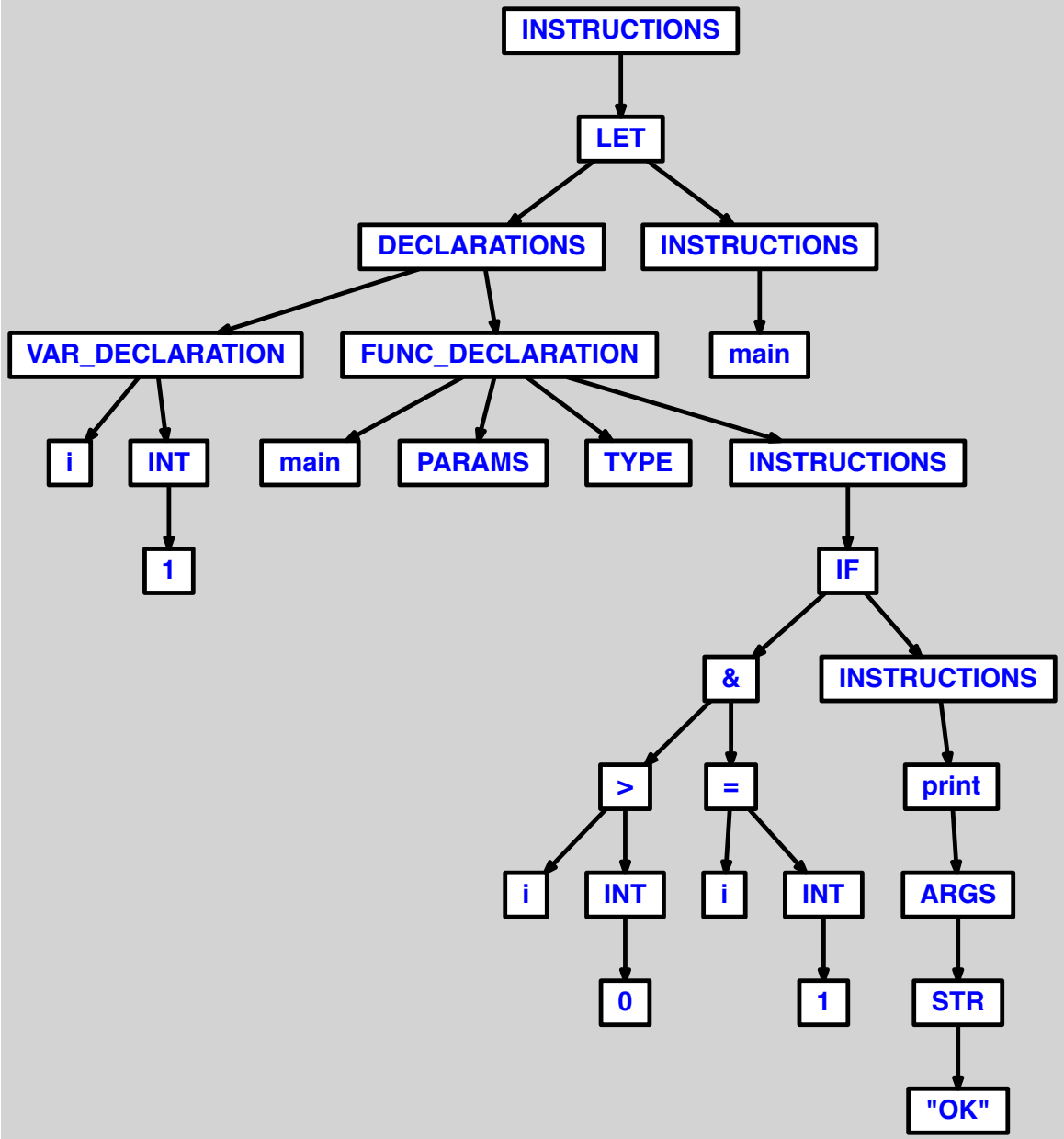
5.2.6 comparaison simple d'entiers avec <"<>">

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


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

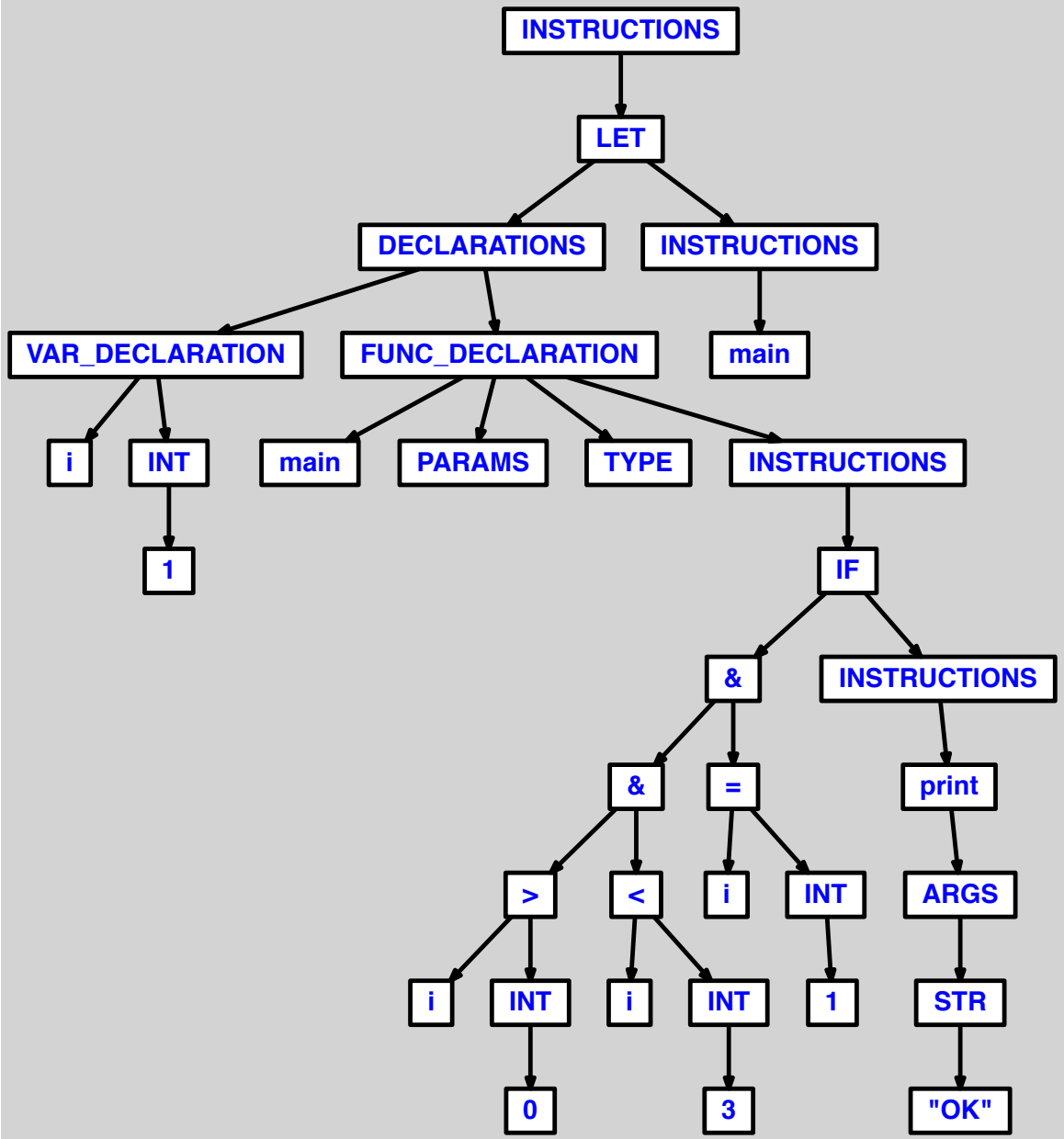
5.2.7 comparaison double d'entiers

```
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
main
i
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
IF
&
INSTRUCTIONS
>
=
print
i
INT
i
INT
ARGS
0
1
STR
"OK"

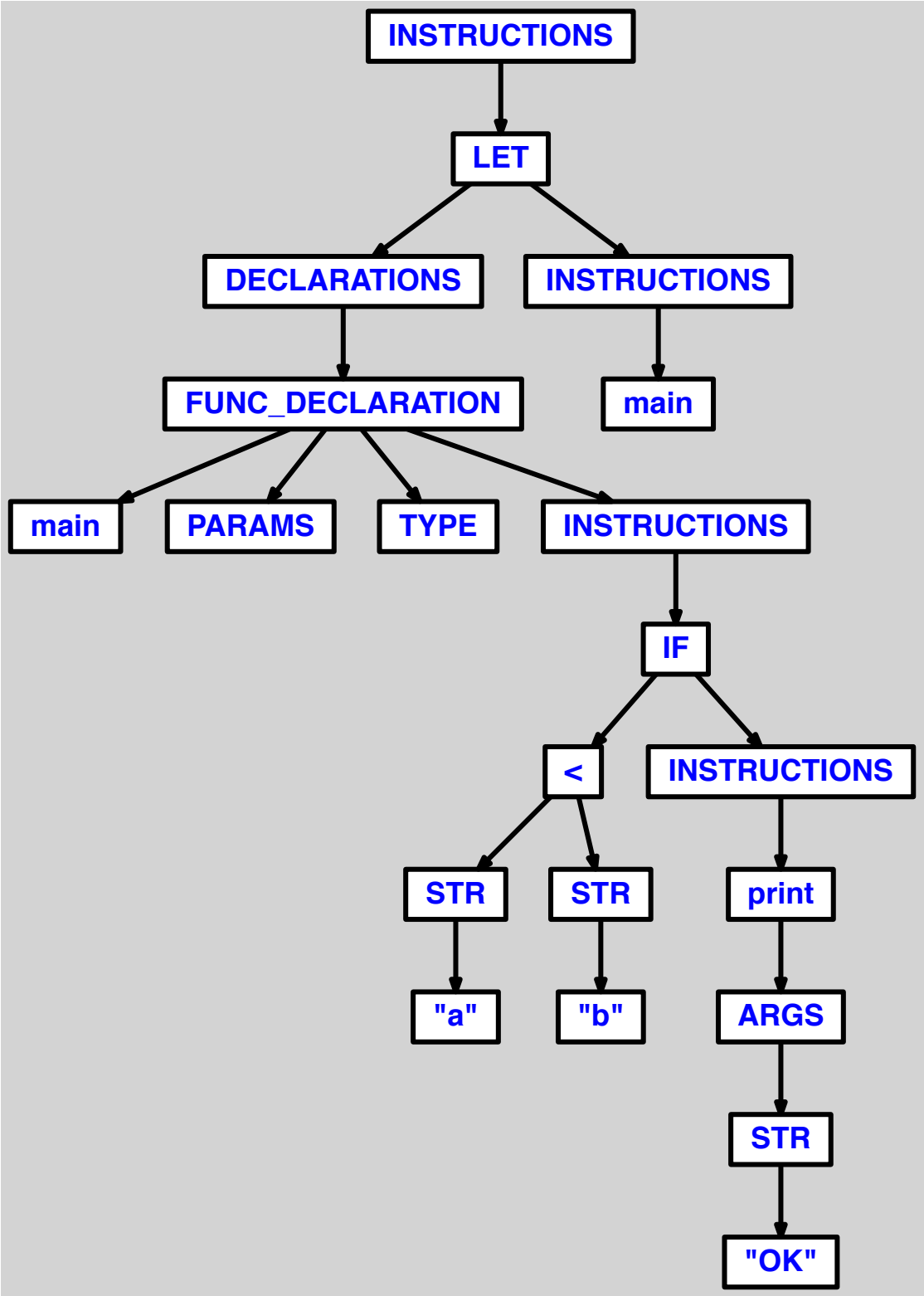
5.2.8 comparaison triple d'entiers

```
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
main
i
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
IF
&
INSTRUCTIONS
&
=
print
>
<
i
INT
ARGS
i
INT
i
INT
1
STR
0
3
"OK"

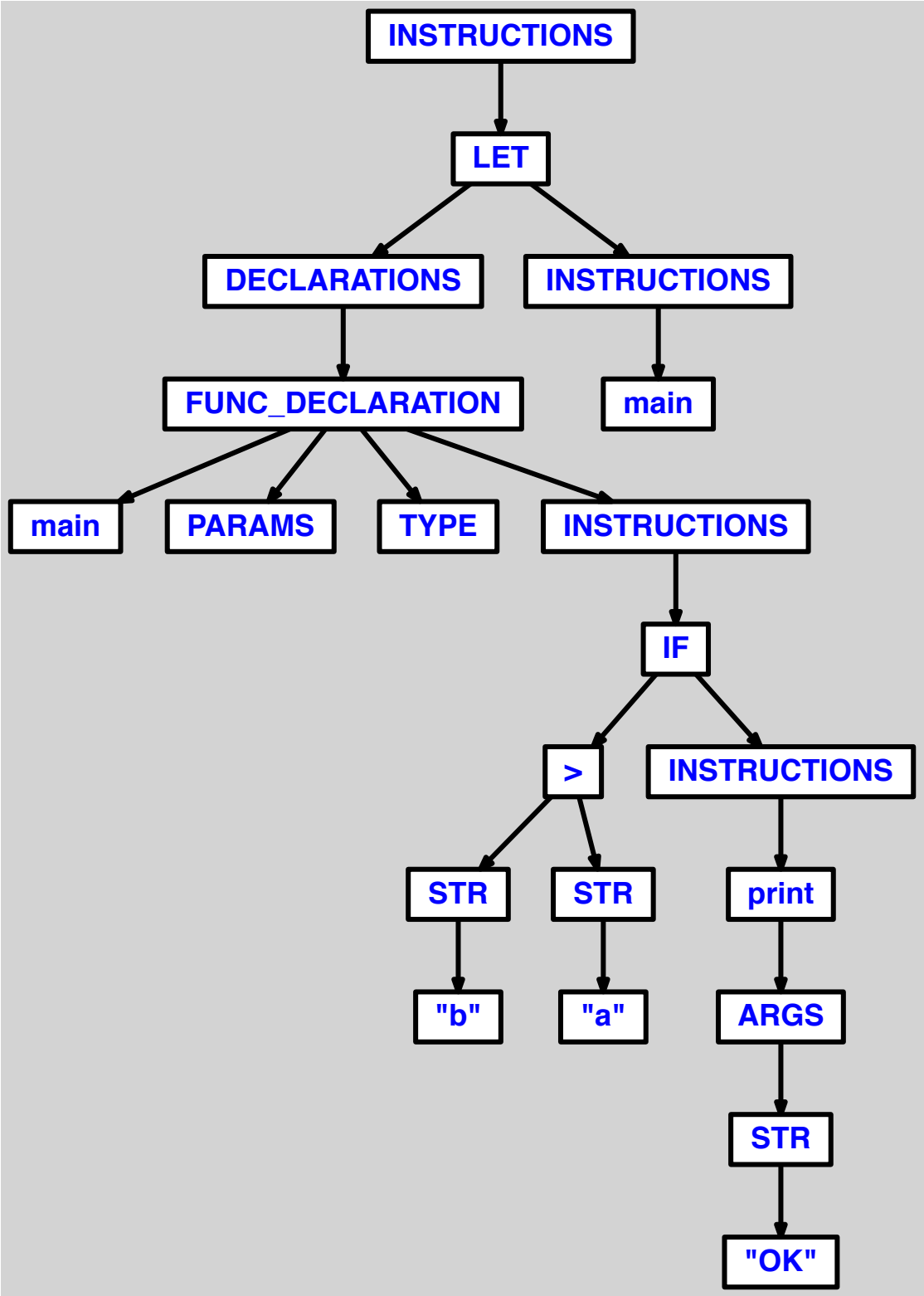
5.2.9 comparaison simple de chaines avec <"<">

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


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

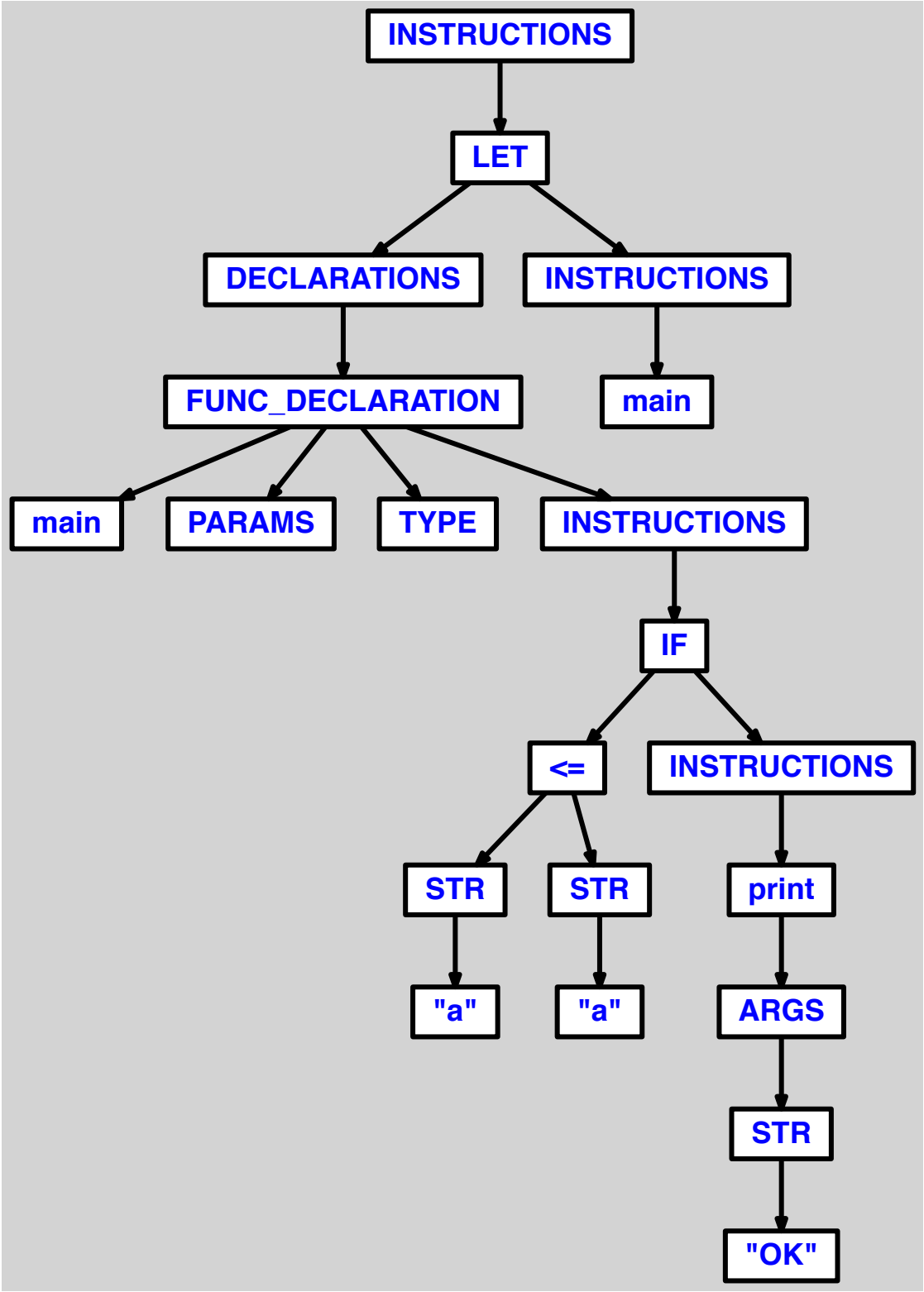
5.2.10 comparaison simple de chaines avec <">">

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


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

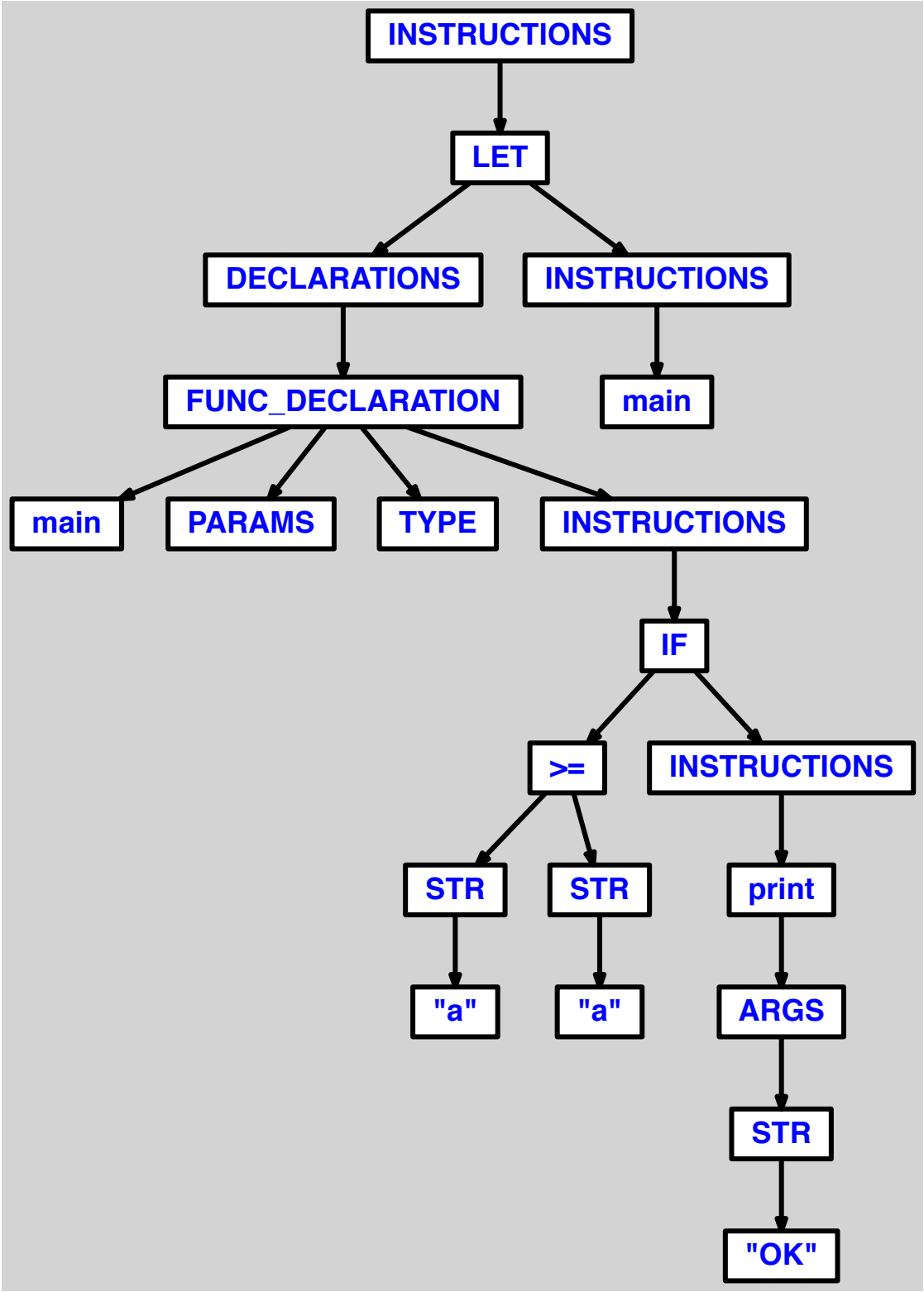
5.2.11 comparaison simple de chaines avec <"<="><

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


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

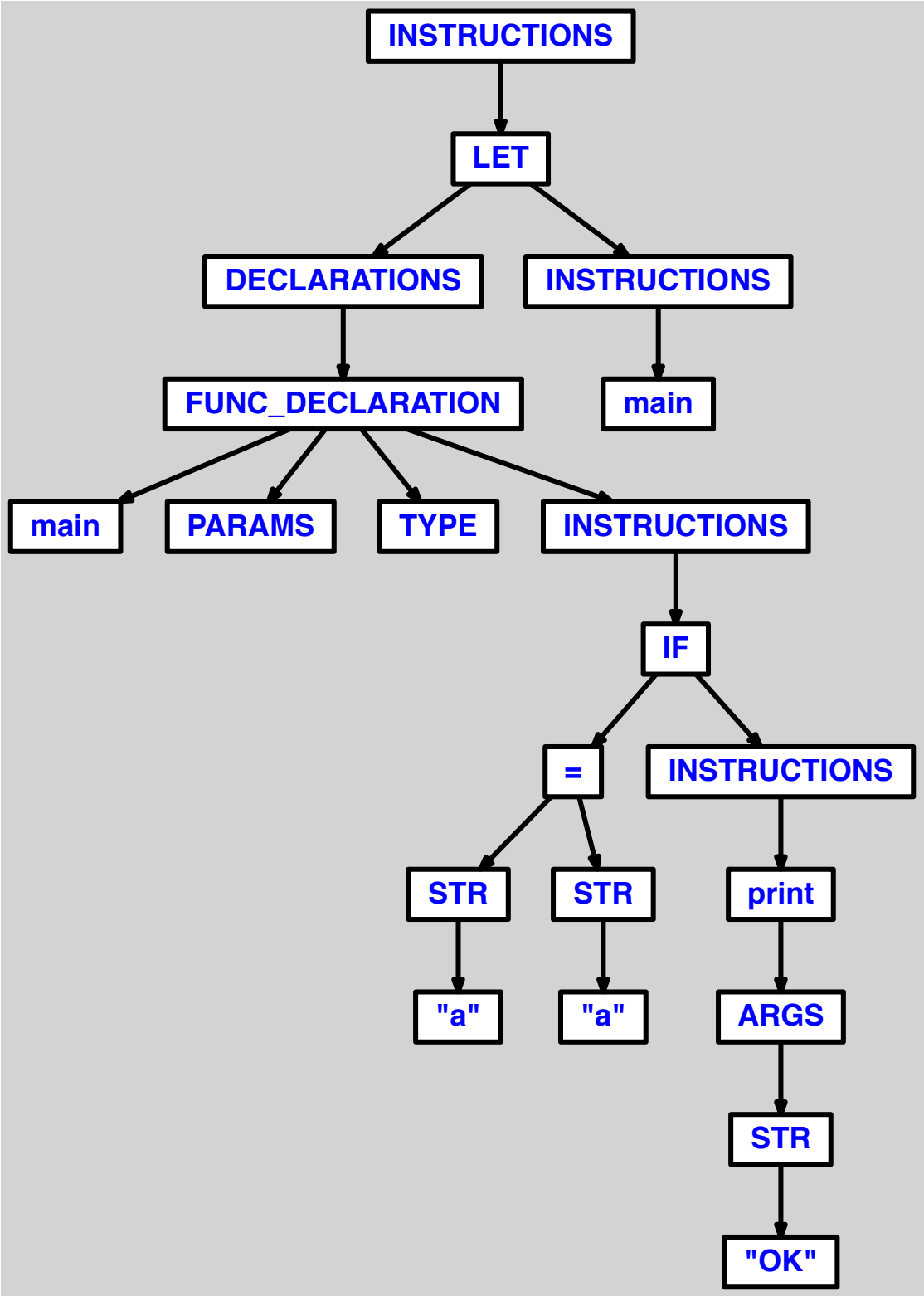
5.2.12 comparaison simple de chaines avec <">="><

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


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

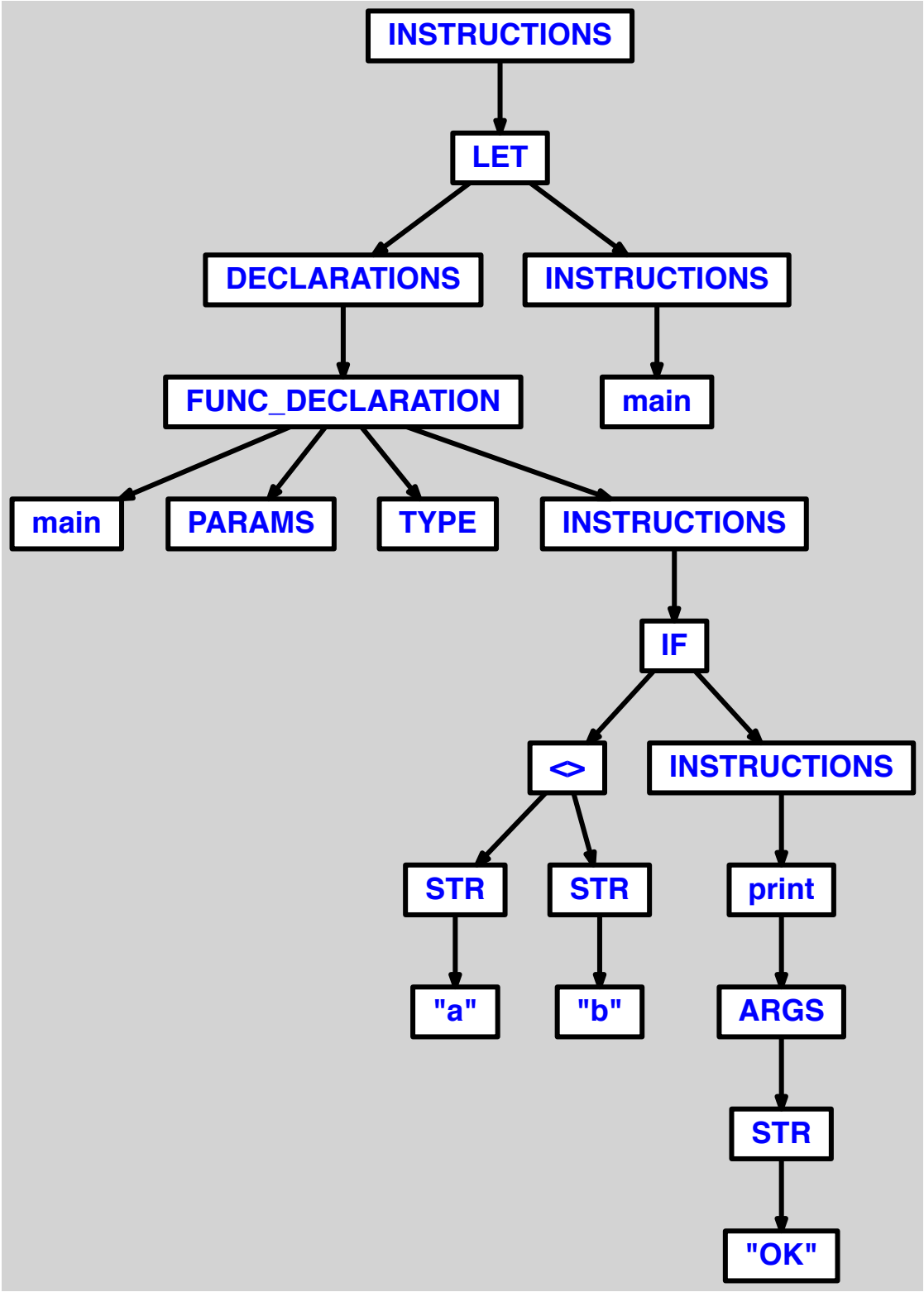
5.2.13 comparaison simple de chaines avec <=>

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


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

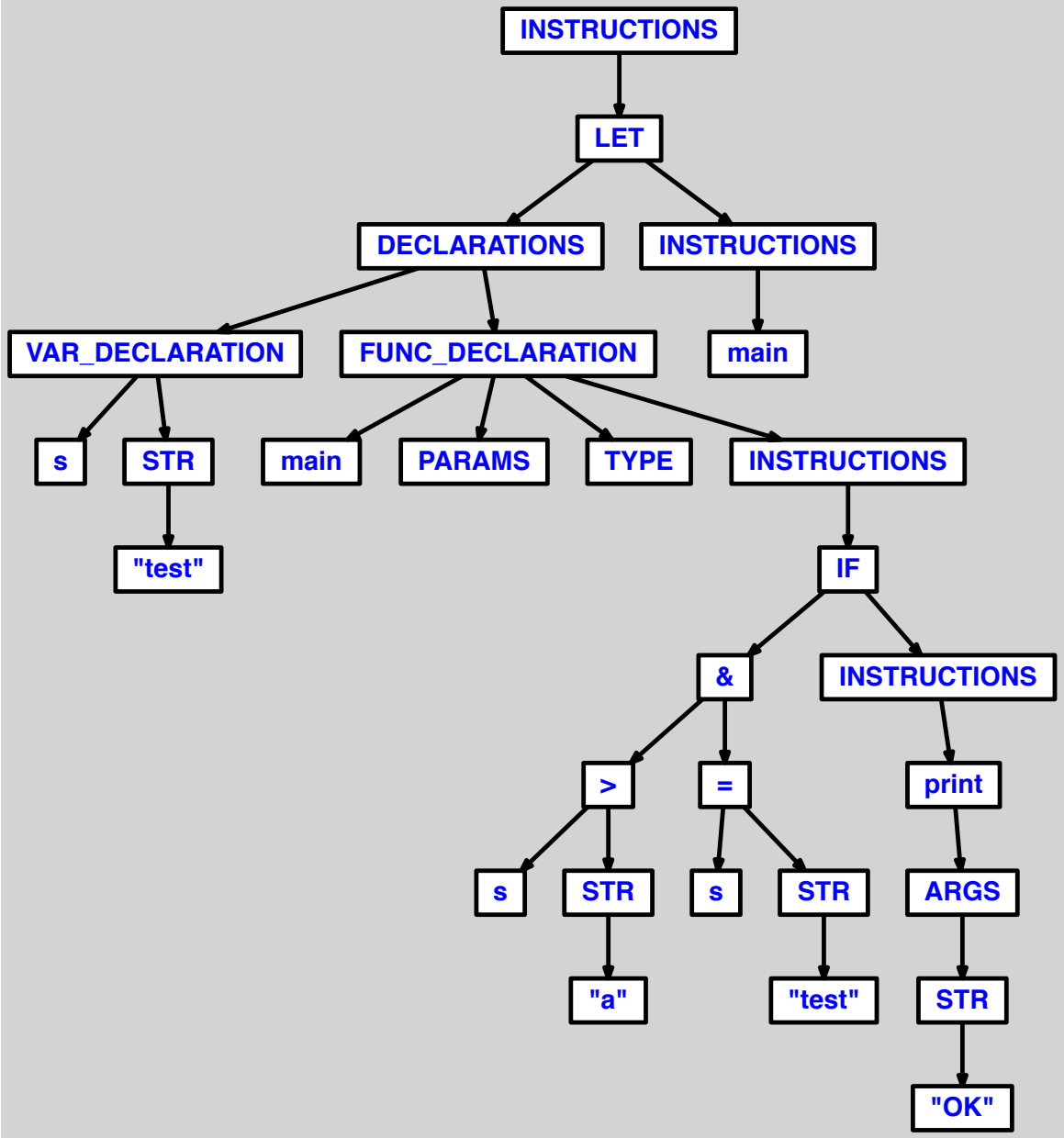
5.2.14 comparaison simple de chaines avec <"<>">

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


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

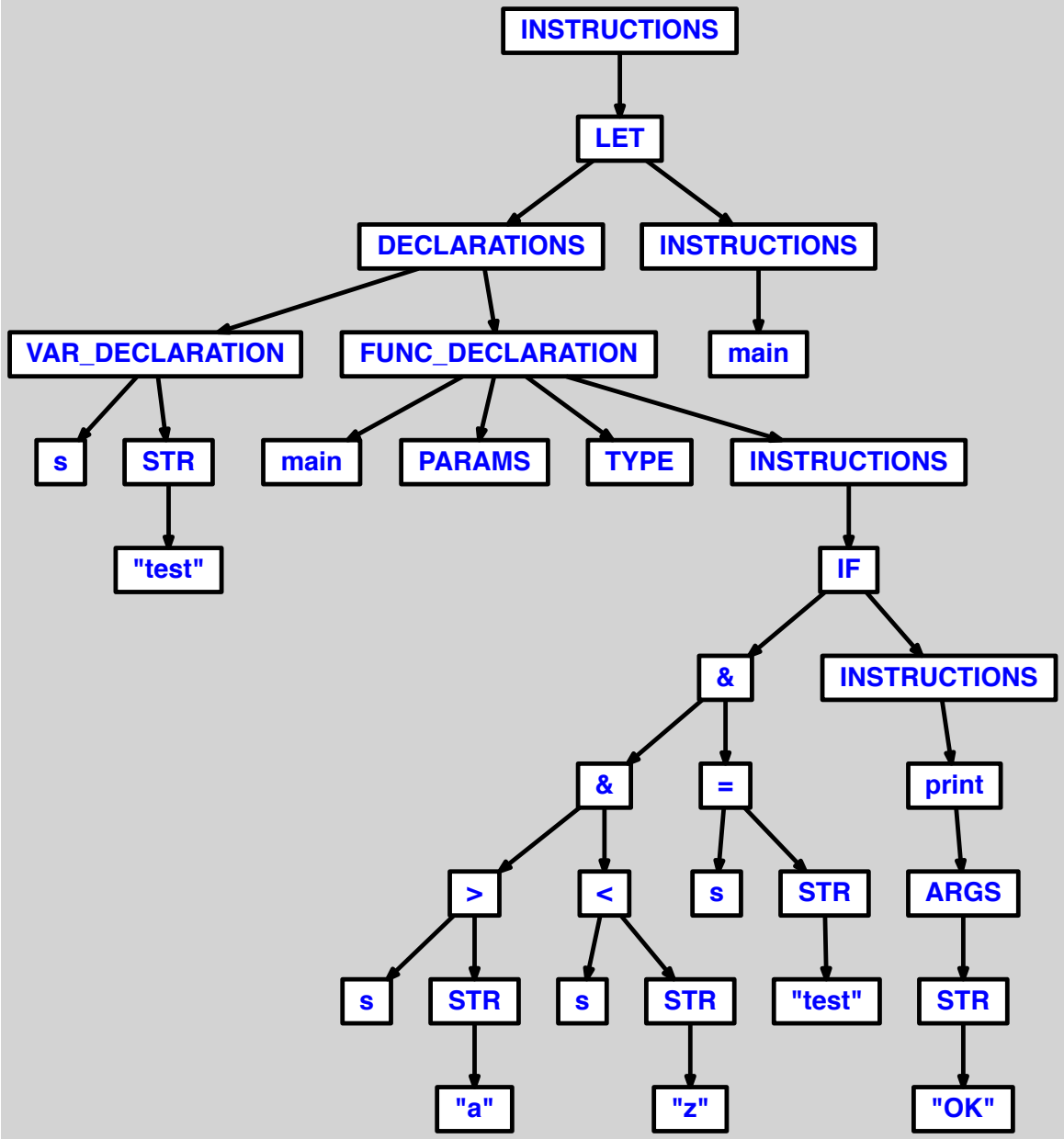
5.2.15 comparaison double de chaines

```
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
main
s
STR
main
PARAMS
TYPE
INSTRUCTIONS
"test"
IF
&
INSTRUCTIONS
>
=
print
s
STR
s
STR
ARGS
"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
main
s
STR
main
PARAMS
TYPE
INSTRUCTIONS
"test"
IF
&
INSTRUCTIONS
&
=
print
>
<
s
STR
ARGS
s
STR
s
STR
"test"
STR
"a"
"z"
"OK"

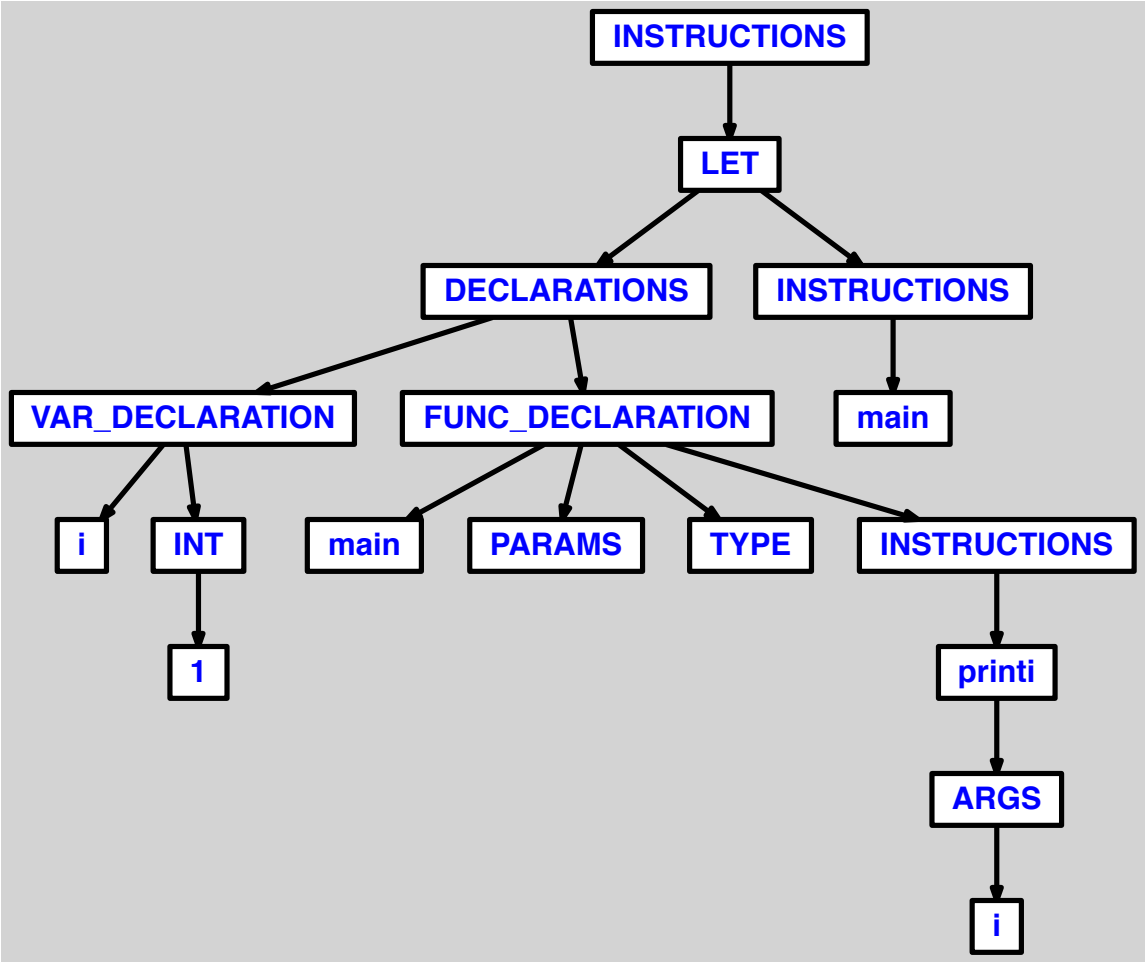
6 espace

6.1 KO

6.2 OK

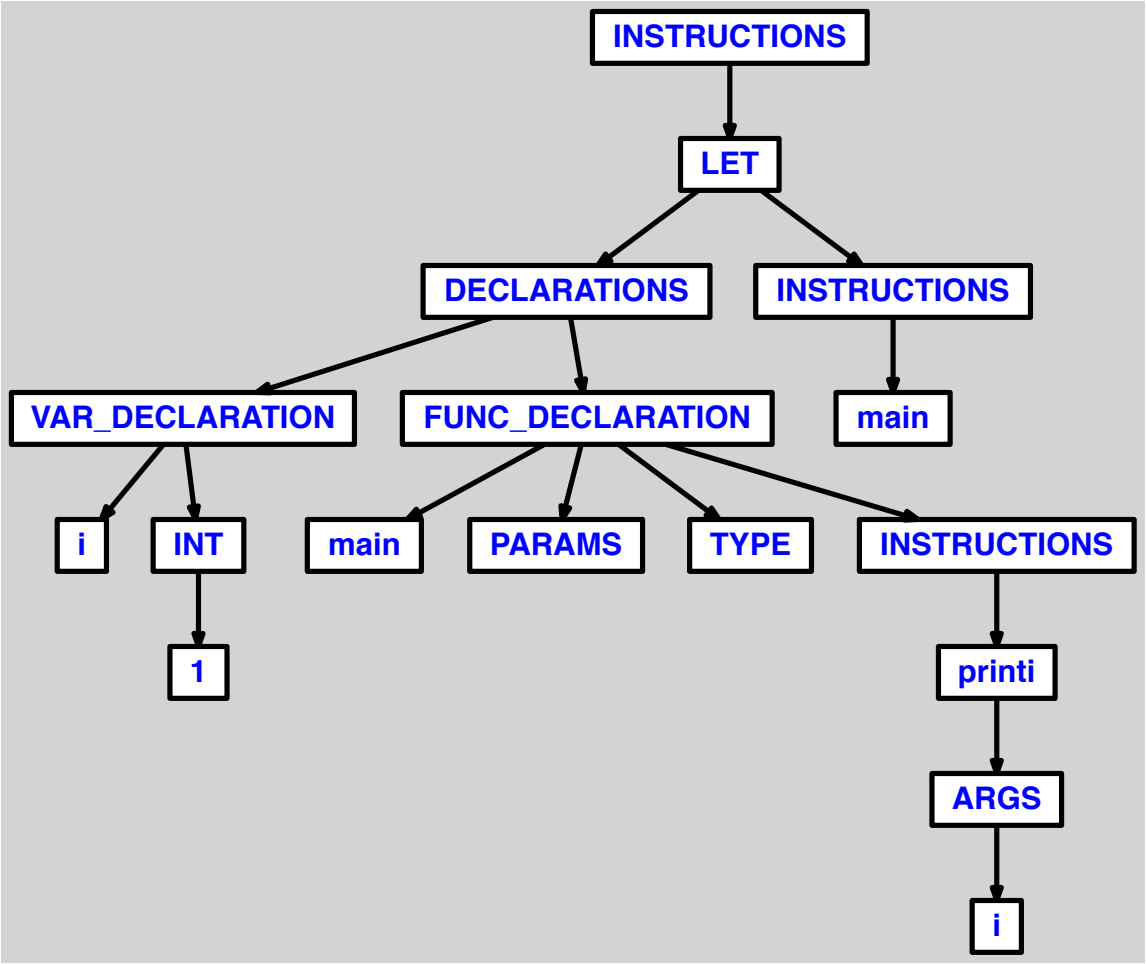
6.2.1 1 espace entre <var> et <i>

```
let
  var i := 1

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


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

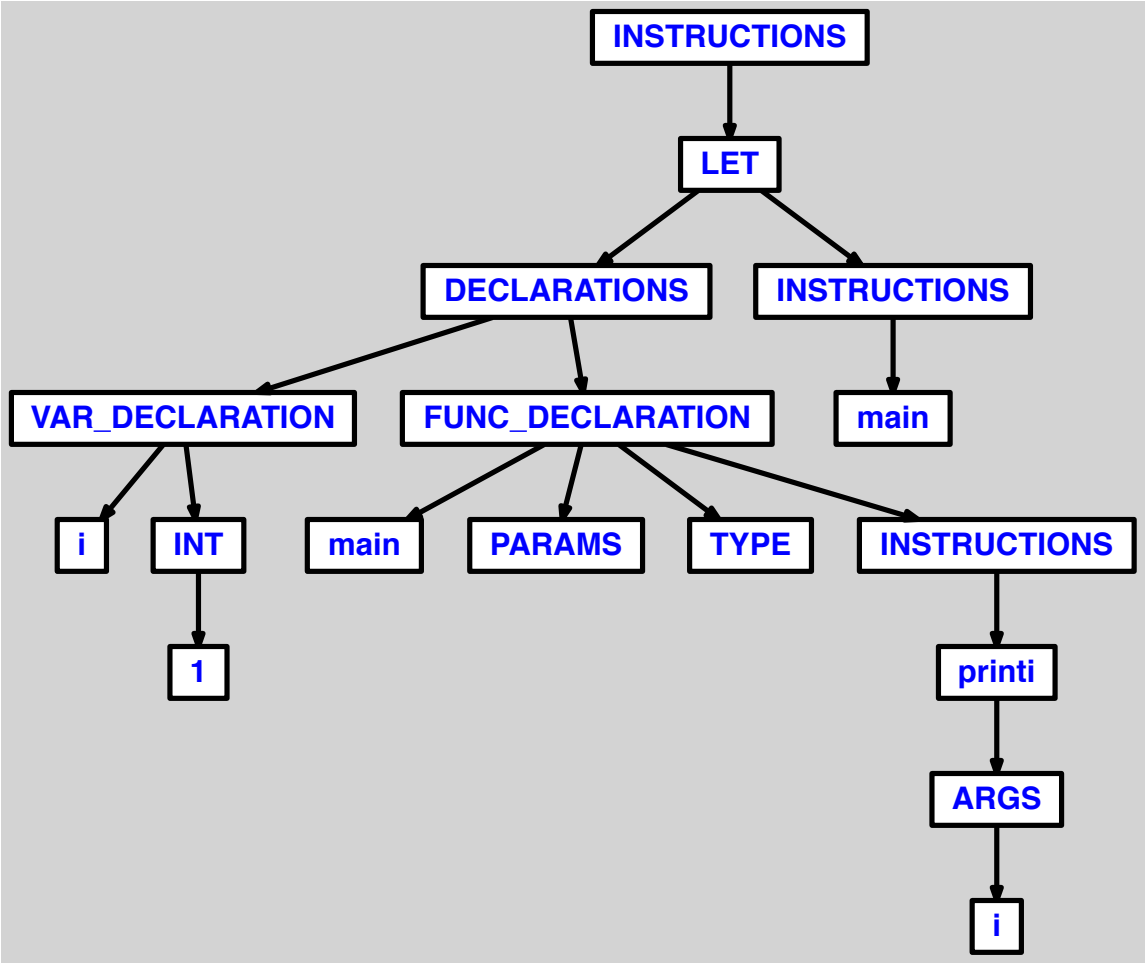
6.2.2 1 tabulation entre <var> et <i>

```
let
  var i := 1

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


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

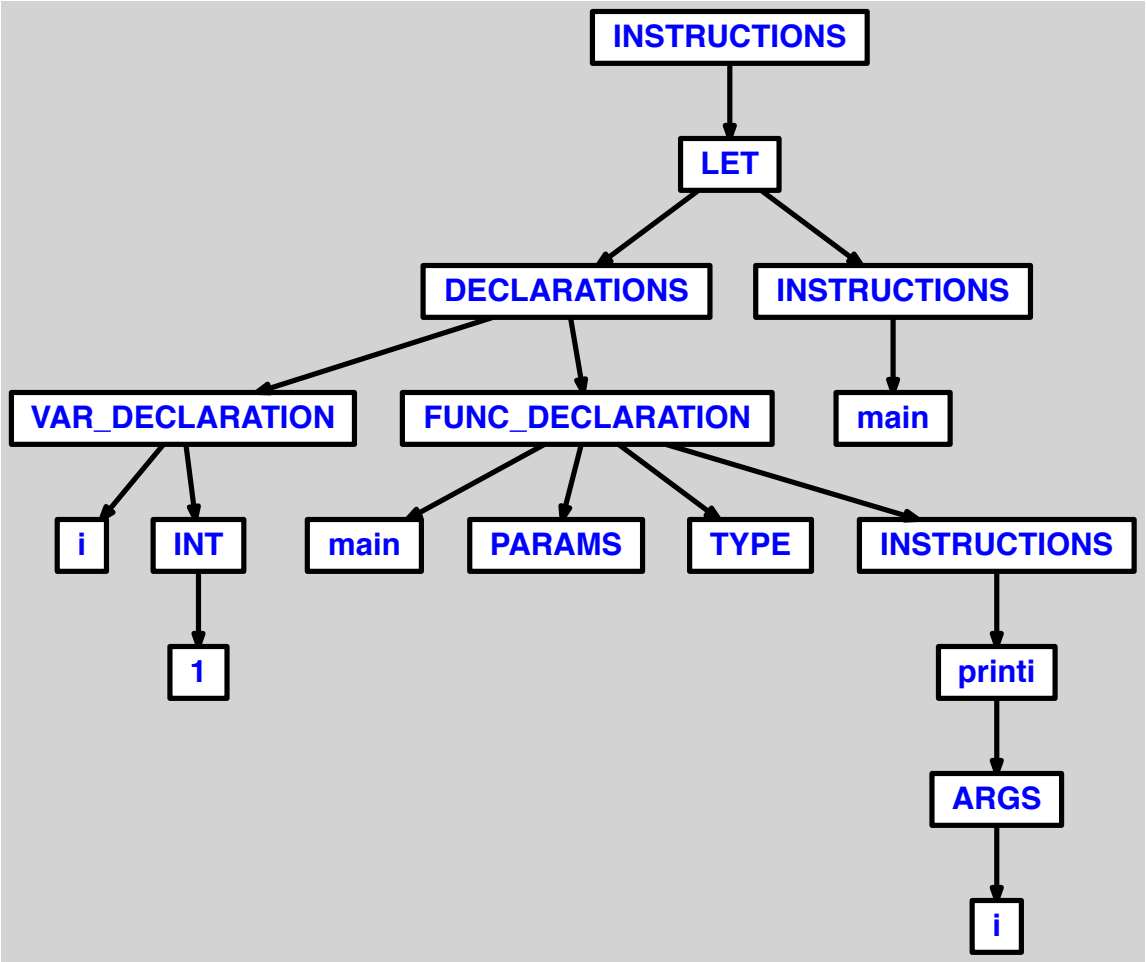
6.2.3 1 saut de ligne entre <var> et <i>

```
let
  var
  i := 1

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


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

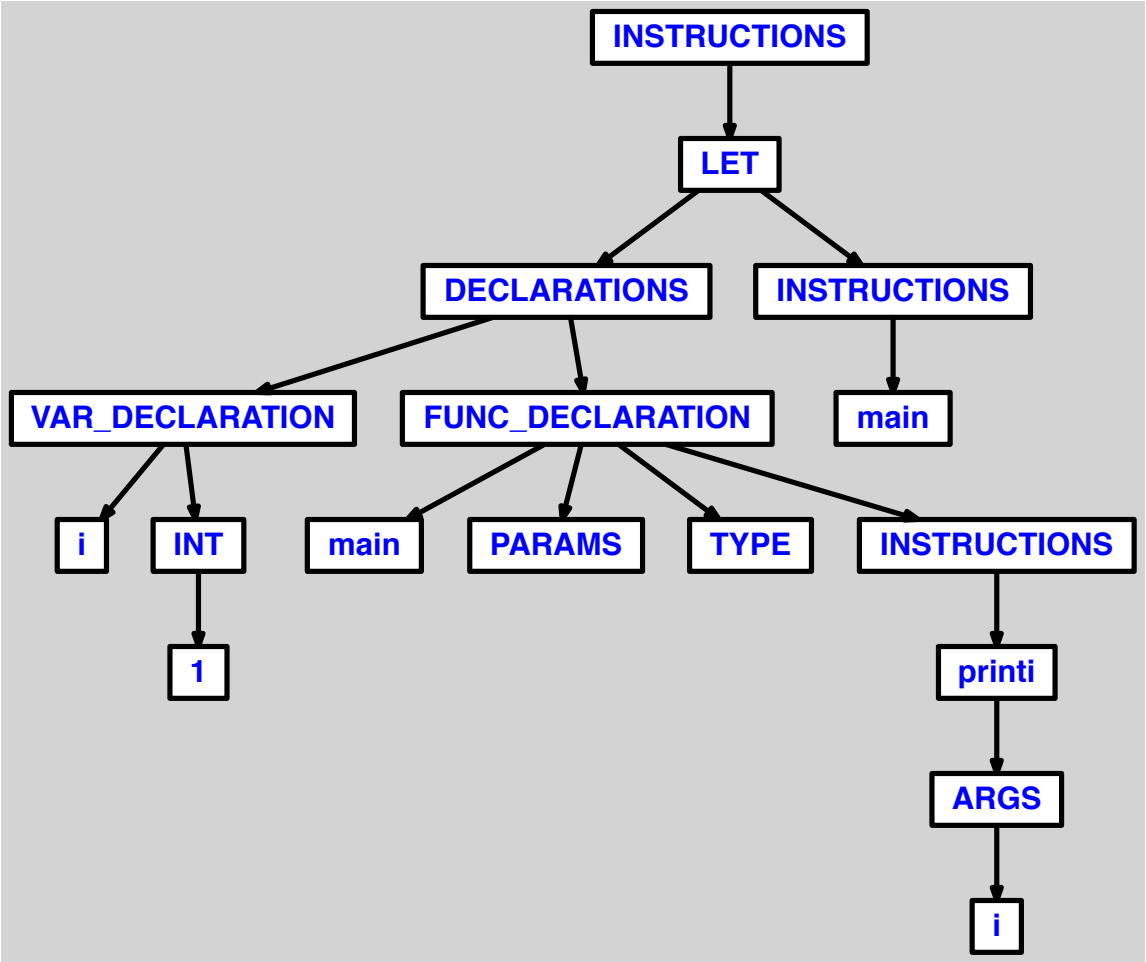
6.2.4 2 espaces entre <var> et <i>

```
let
  var  i := 1

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


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

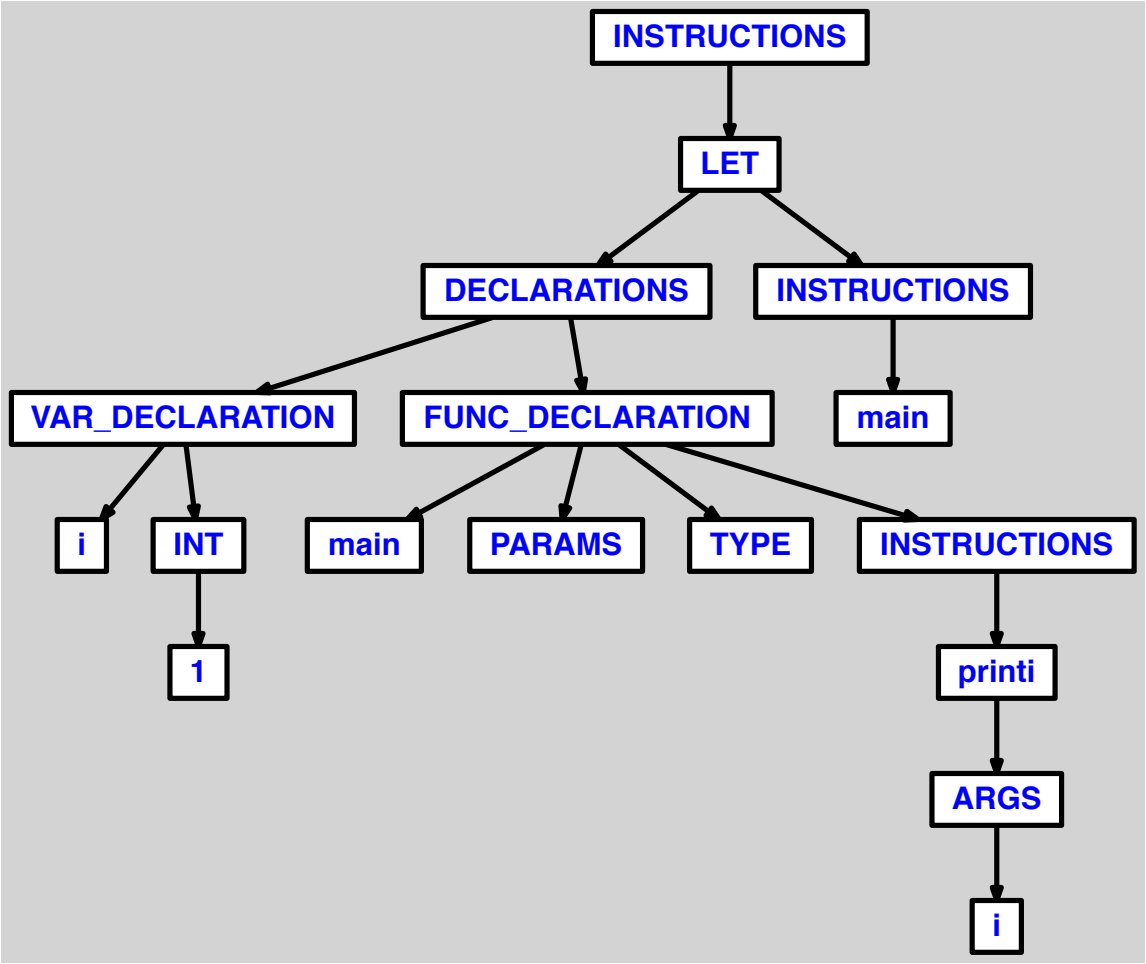
6.2.5 2 tabulations entre <var> et <i>

```
let
  var   i := 1

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


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

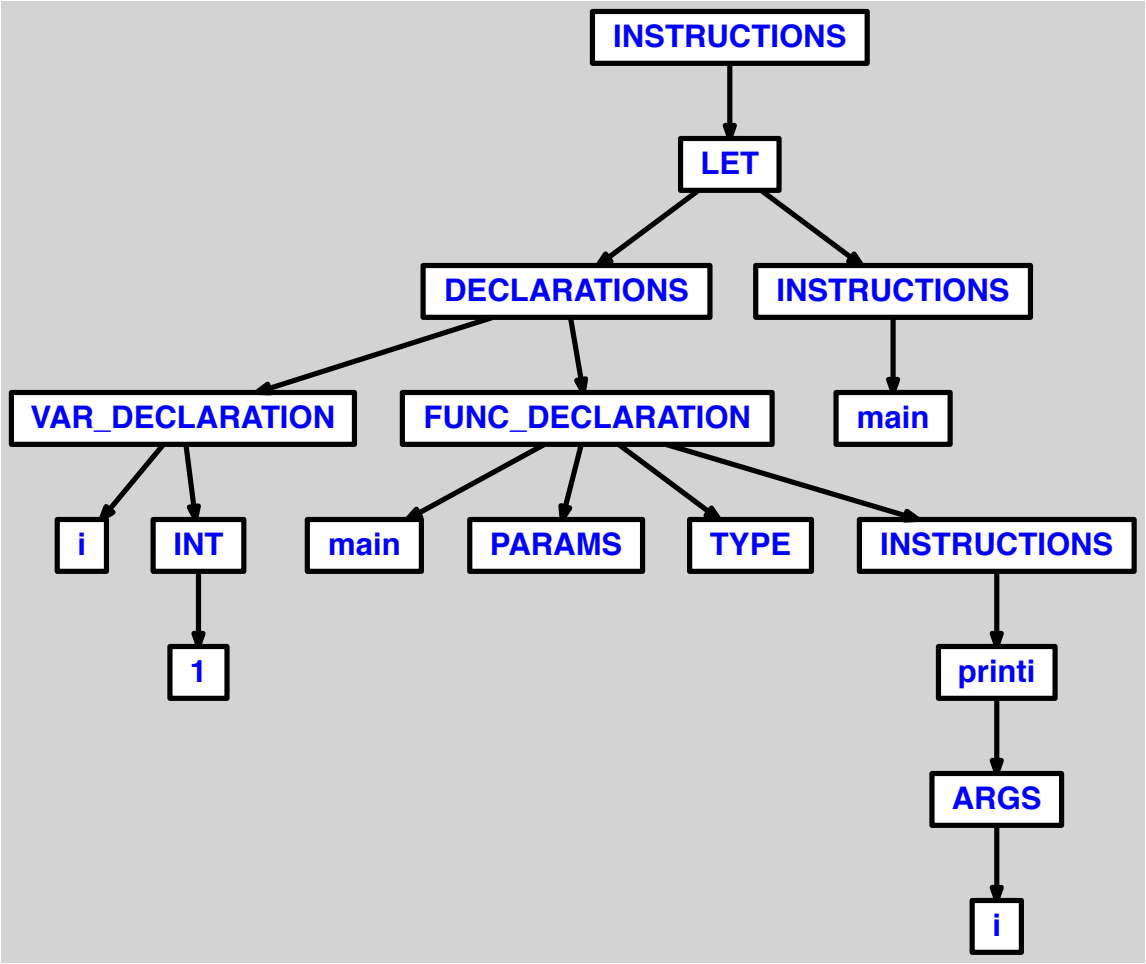
6.2.6 2 sauts de ligne entre <var> et <i>

```
let
  var

  i := 1

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


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

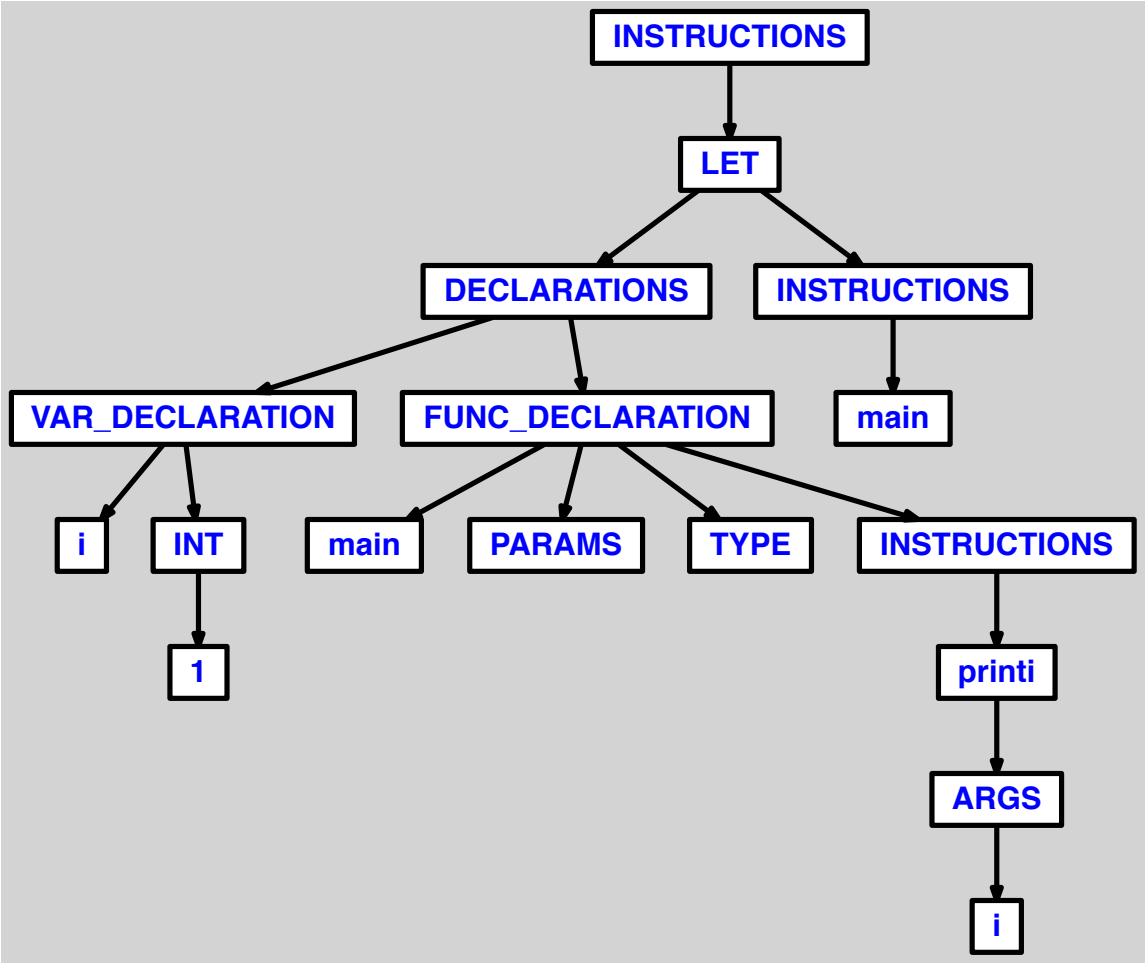
6.2.7 3 espaces entre <var> et <i>

```
let
  var   i := 1

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


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

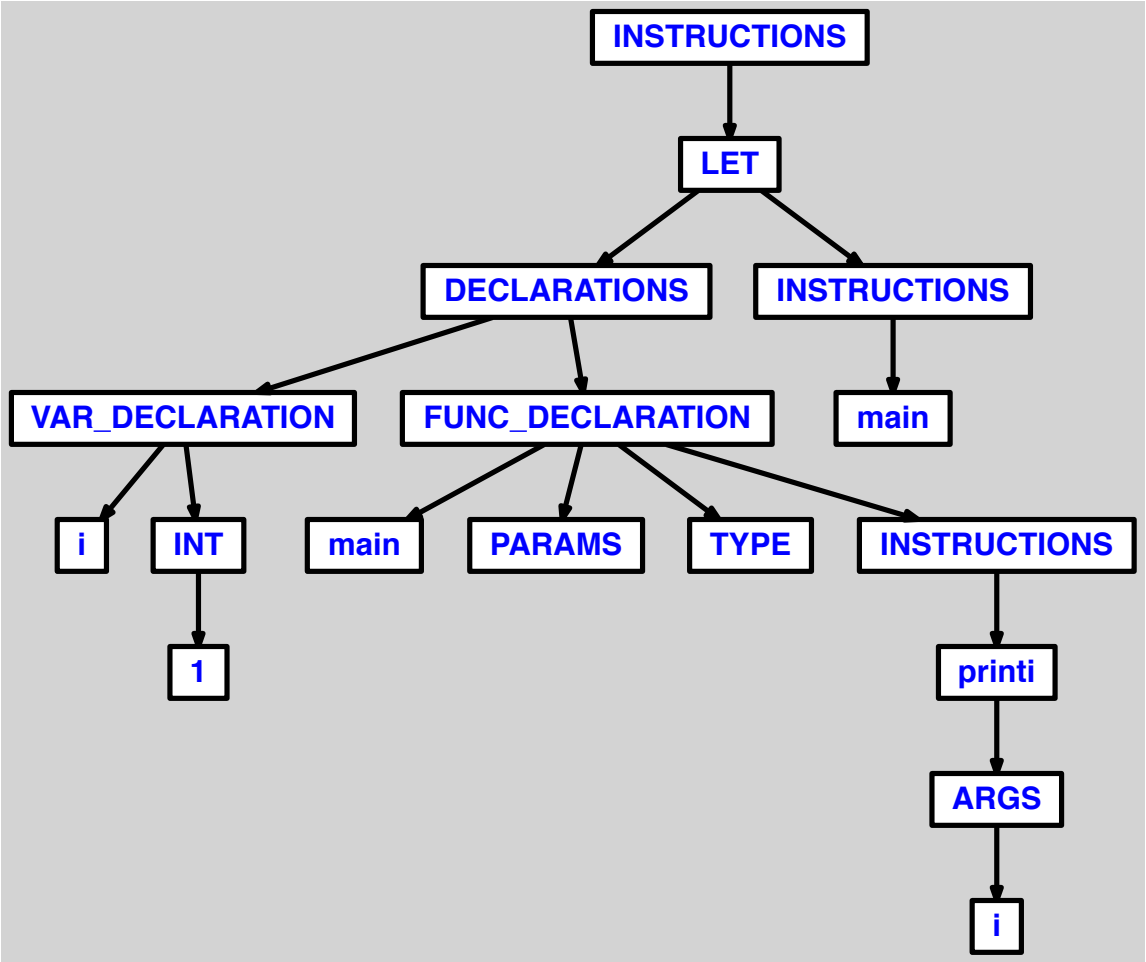
6.2.8 3 tabulations entre <var> et <i>

```
let
  var     i := 1

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


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

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

```
let
  var


  i := 1

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

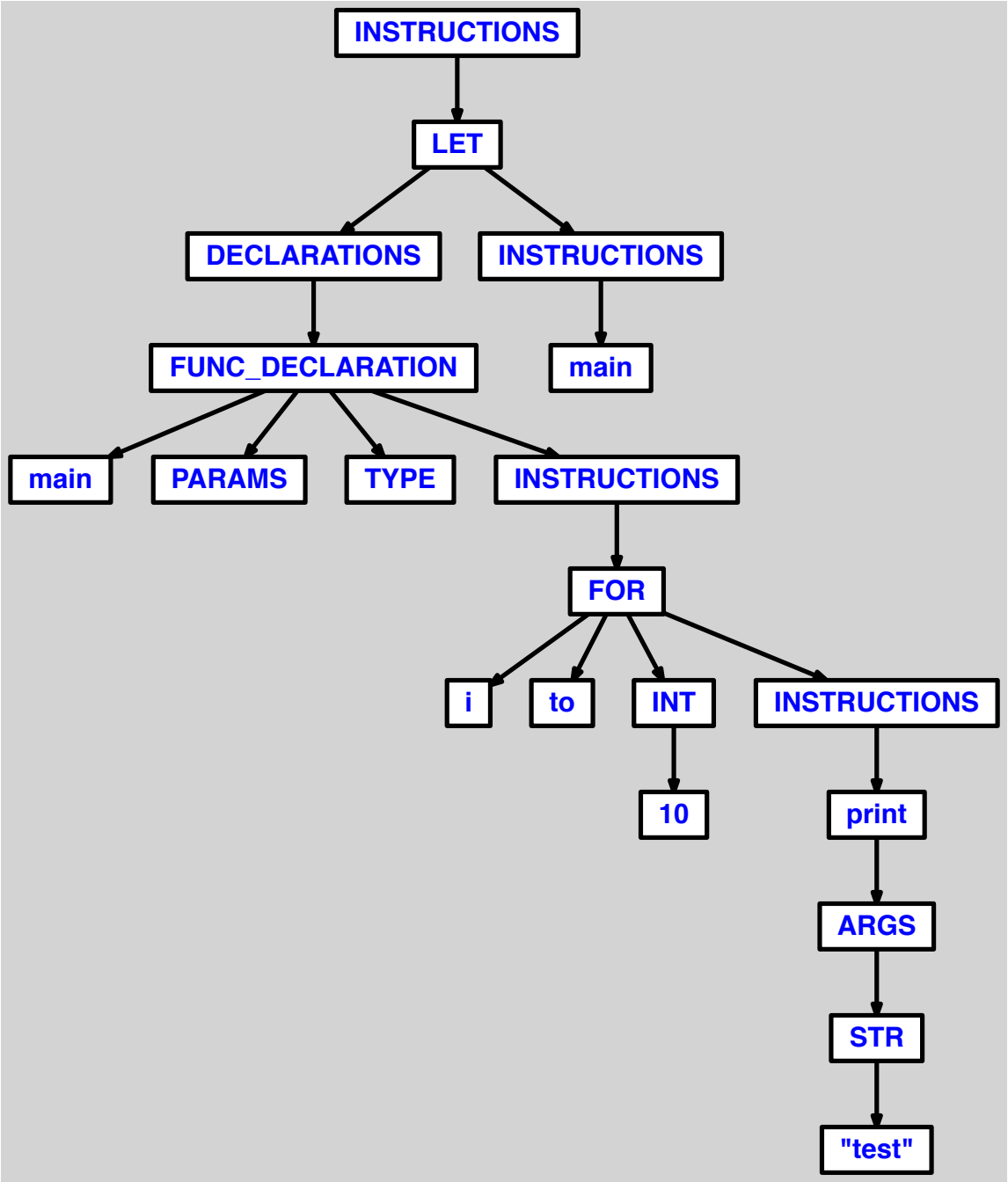

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
main
i
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
printi
ARGS
i

7 for

7.1 KO

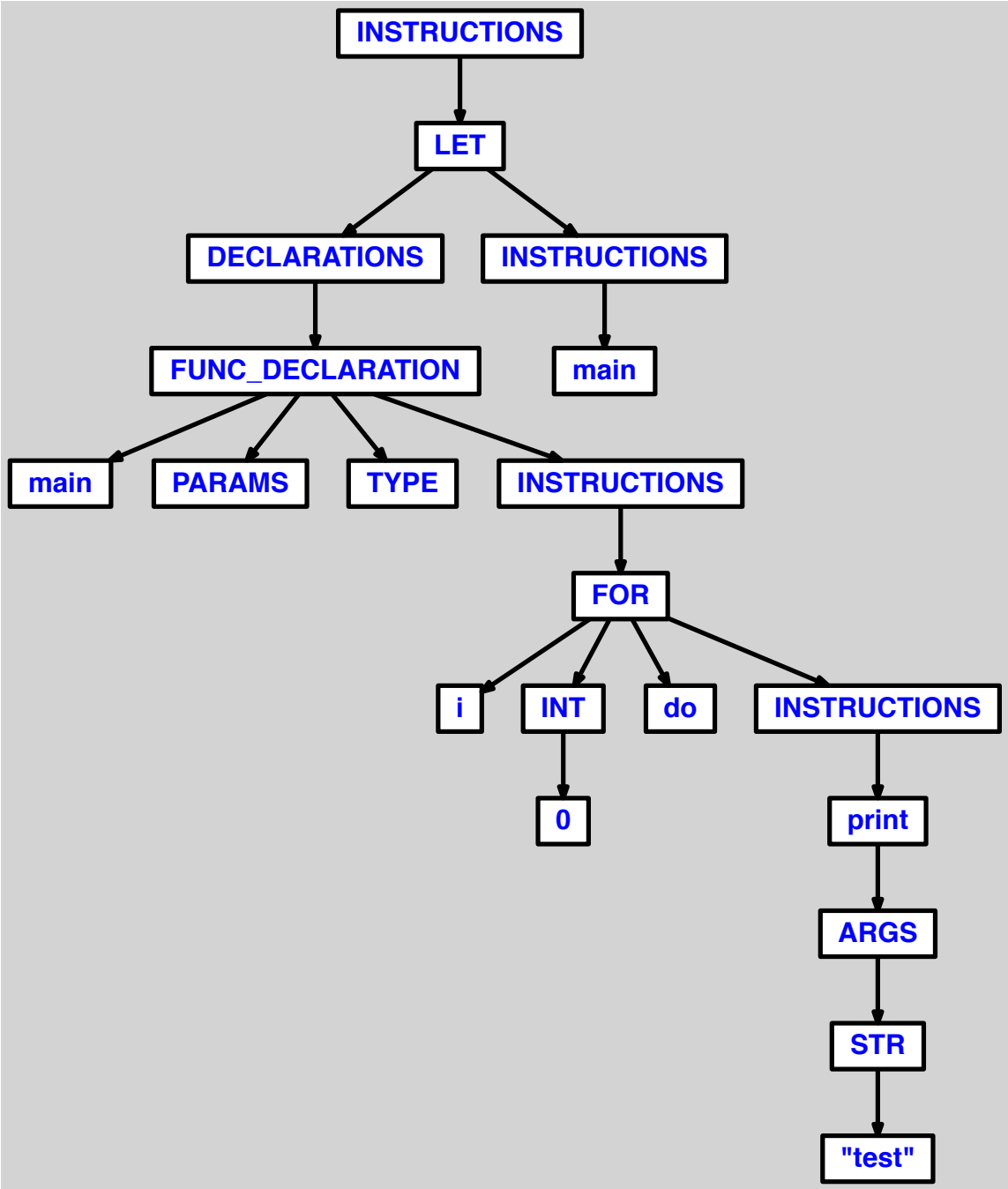
7.1.1 <for> avec affectation de chaine pour le compteur

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
FOR
i
to
INT
INSTRUCTIONS
10
print
ARGS
STR
"test"

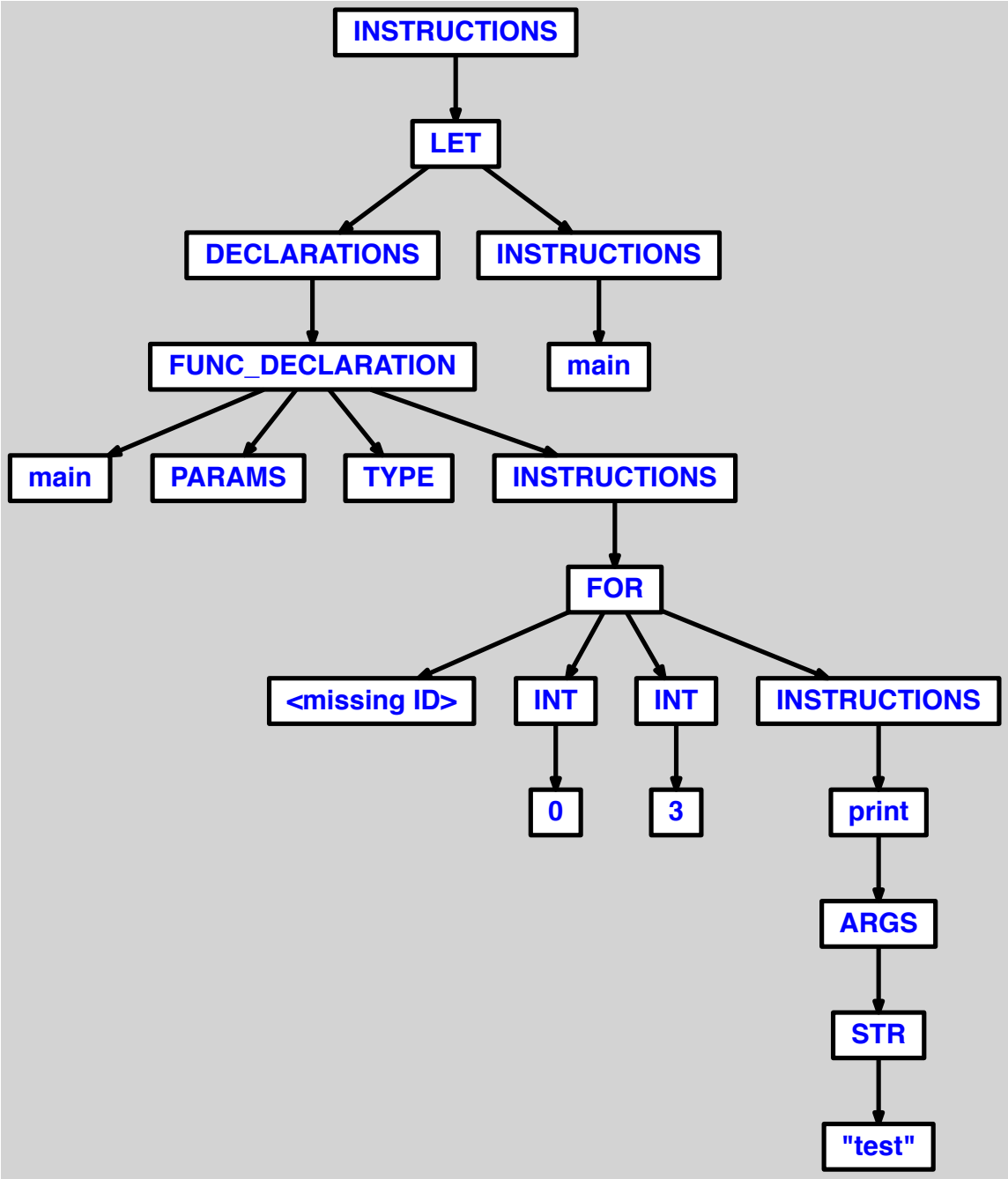
7.1.2 <for> avec affectation de chaine pour la borne superieure de l'iteration

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


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

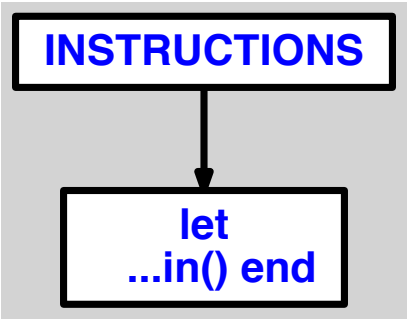
7.1.3 <for> avec oubli de l'identifiant du compteur

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
FOR
<missing ID>
INT
INT
INSTRUCTIONS
0
3
print
ARGS
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

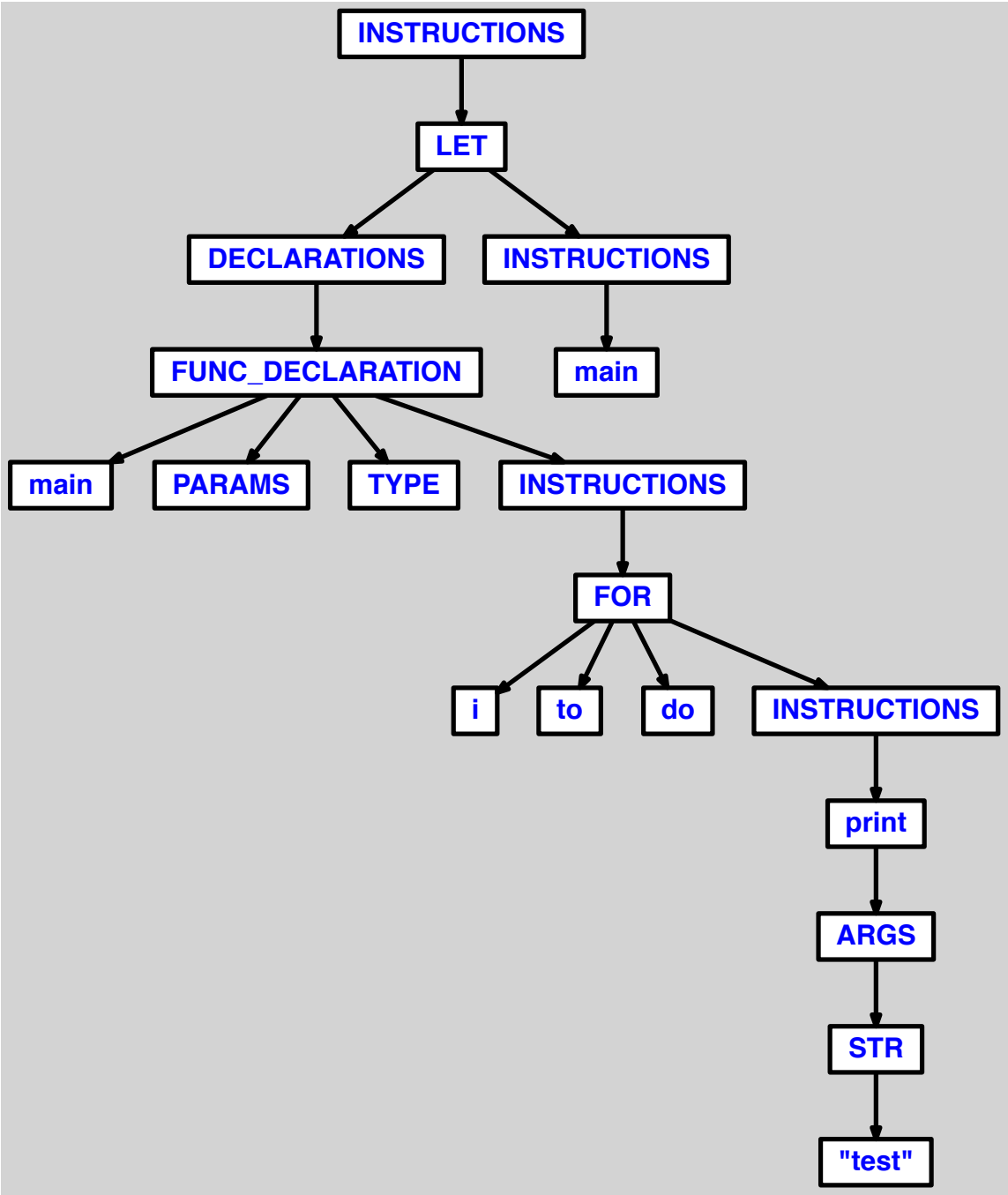
7.1.5 <for> "express"

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

Pas d'AST, problème de syntaxe.

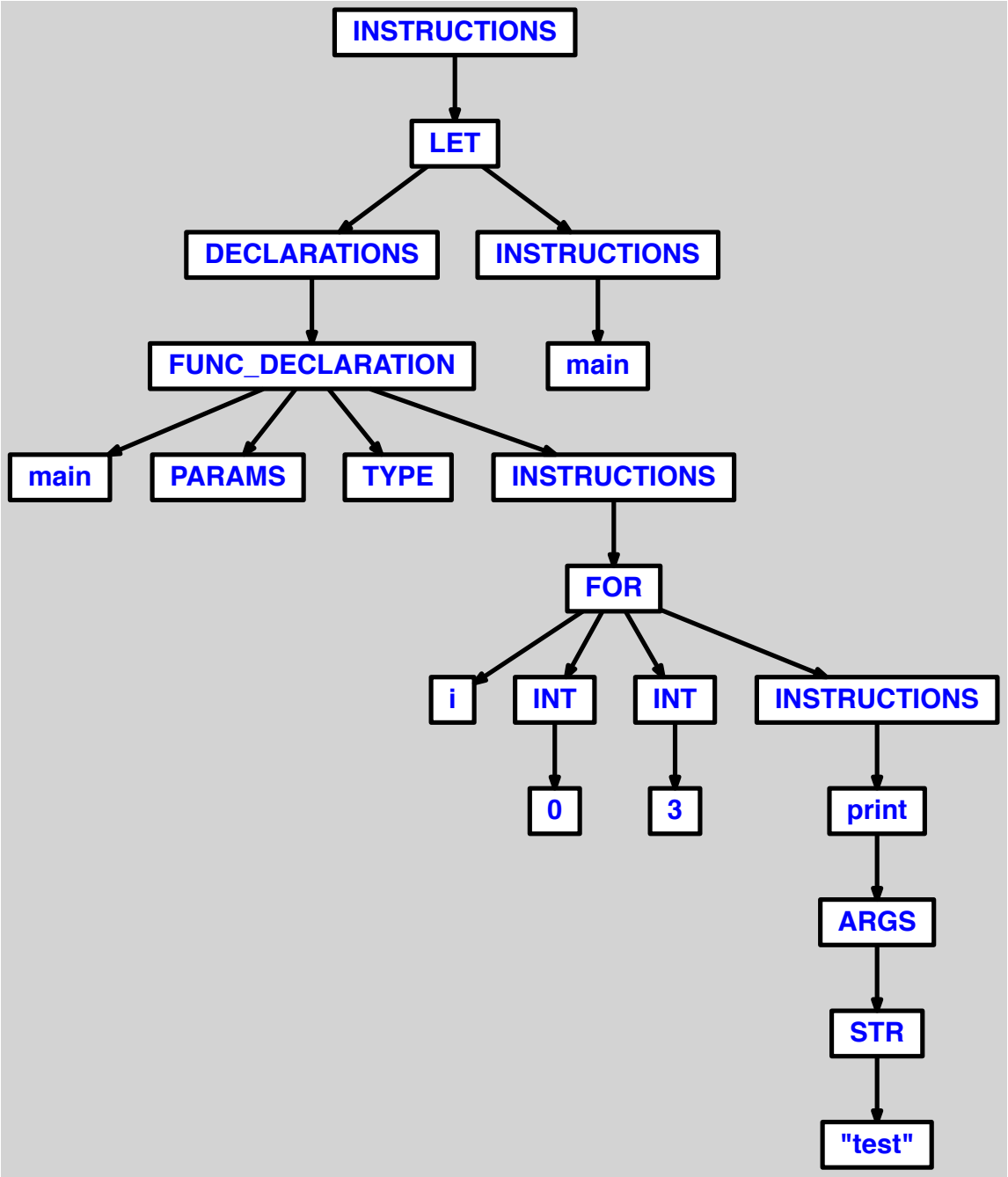
7.1.6 <for> avec affectation de chaines pour le compteur et la borne maximale de l'iteration

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
FOR
i
to
do
INSTRUCTIONS
print
ARGS
STR
"test"

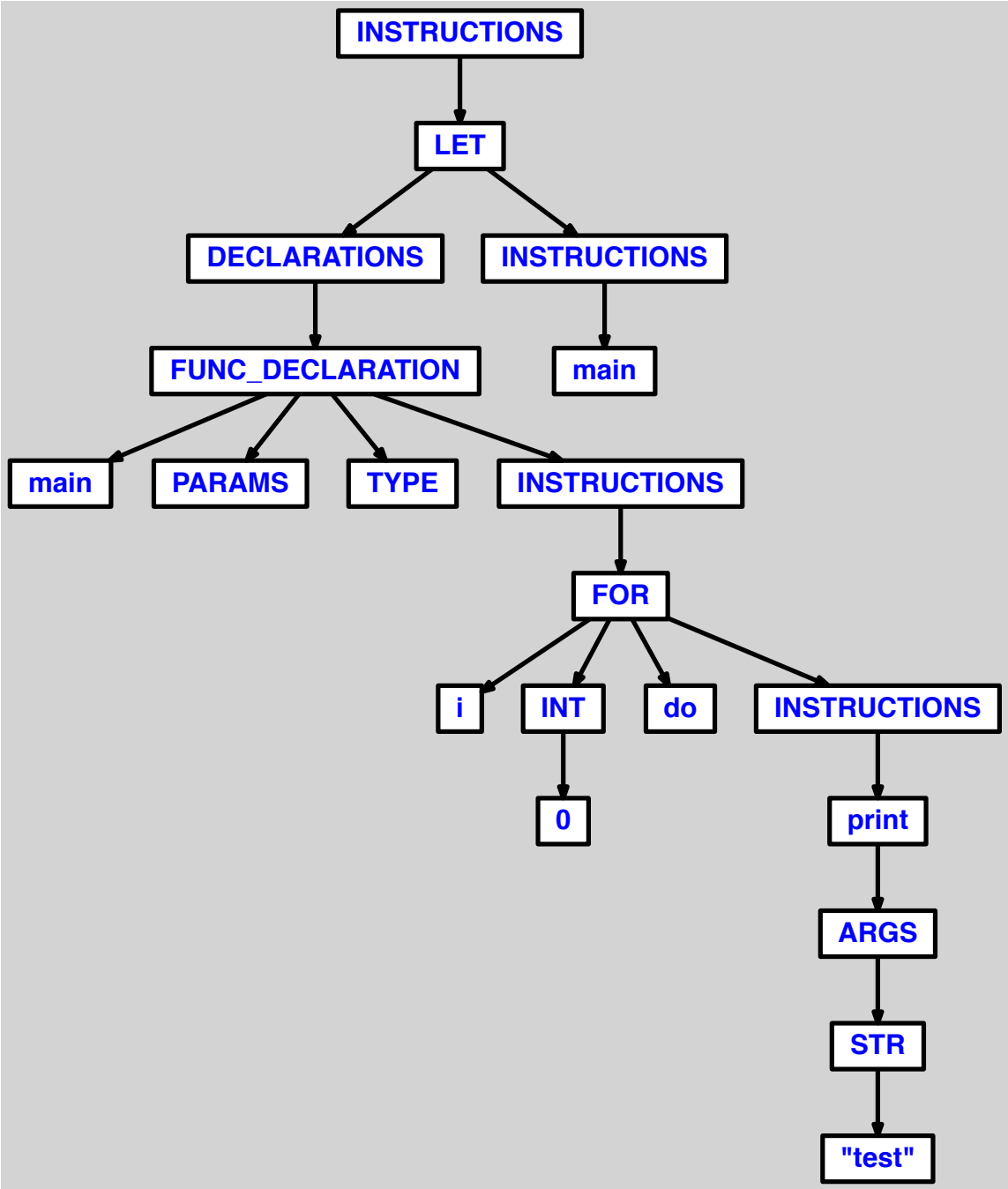
7.1.7 <for> avec oubli du <do>

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


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

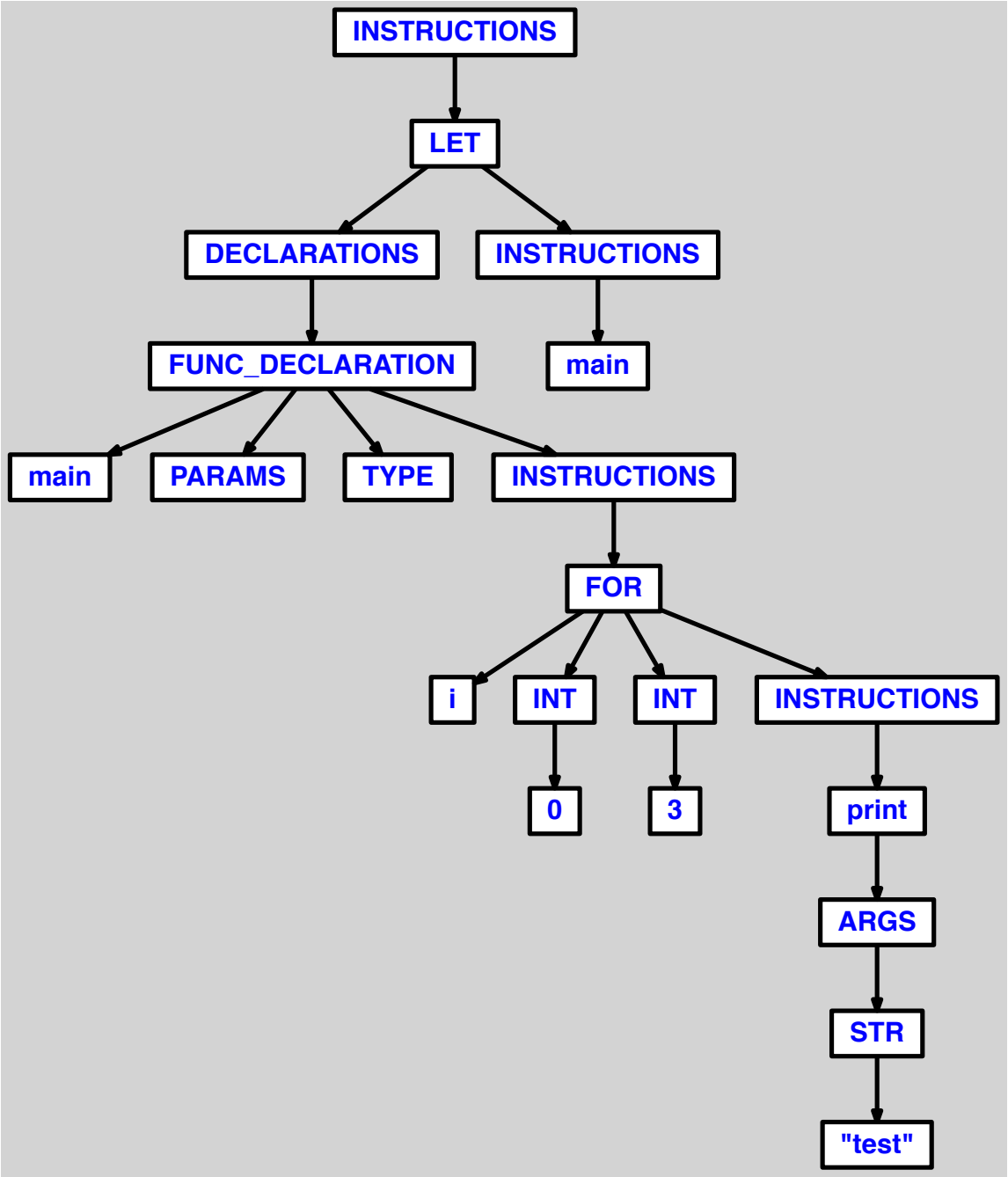
7.1.8 <for> avec oubli de la borne maximale de l'iteration

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


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

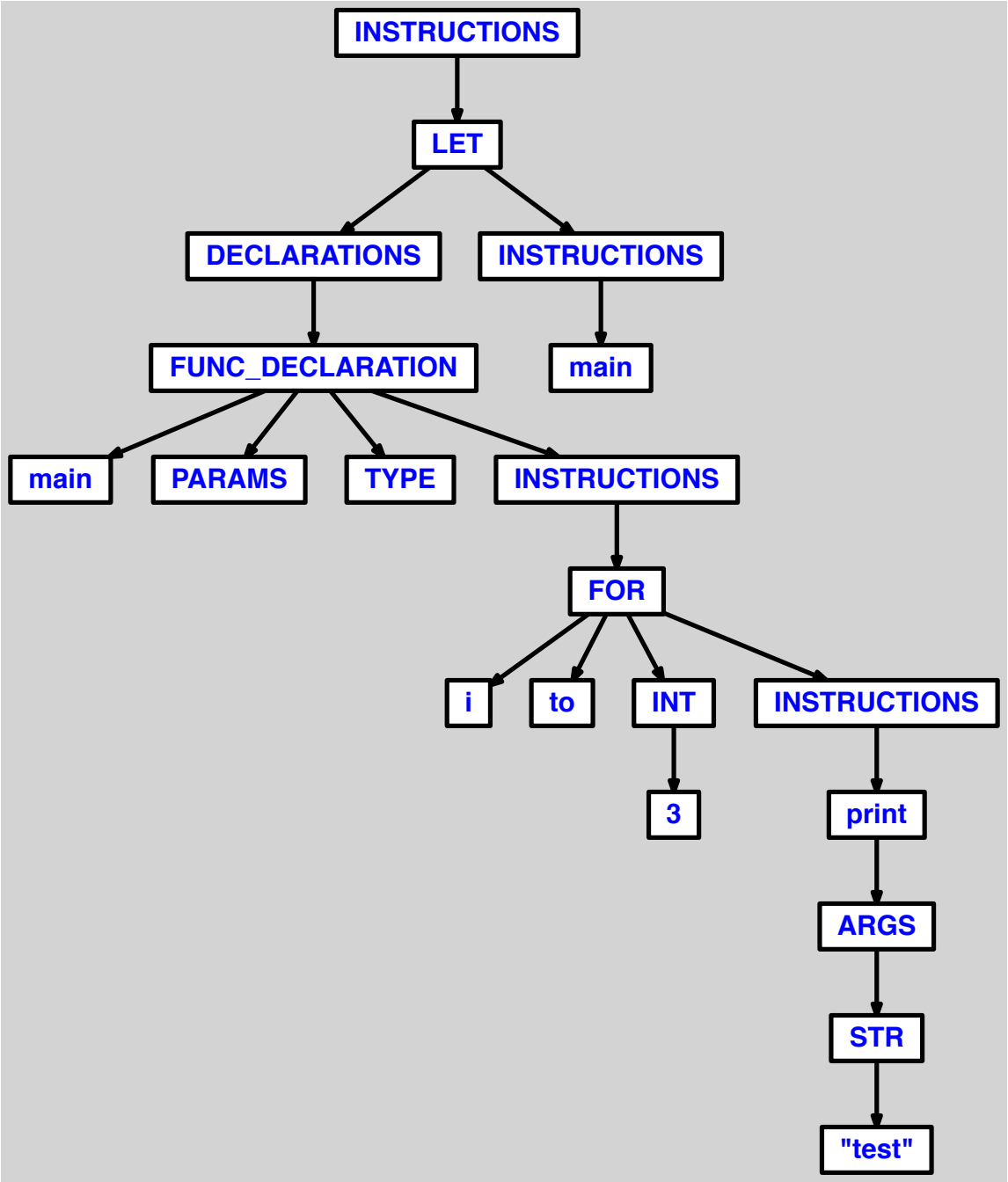
7.1.9 <for> avec oubli du <to>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
FOR
i
INT
INT
INSTRUCTIONS
0
3
print
ARGS
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
main
main
PARAMS
TYPE
INSTRUCTIONS
FOR
i
to
INT
INSTRUCTIONS
3
print
ARGS
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.

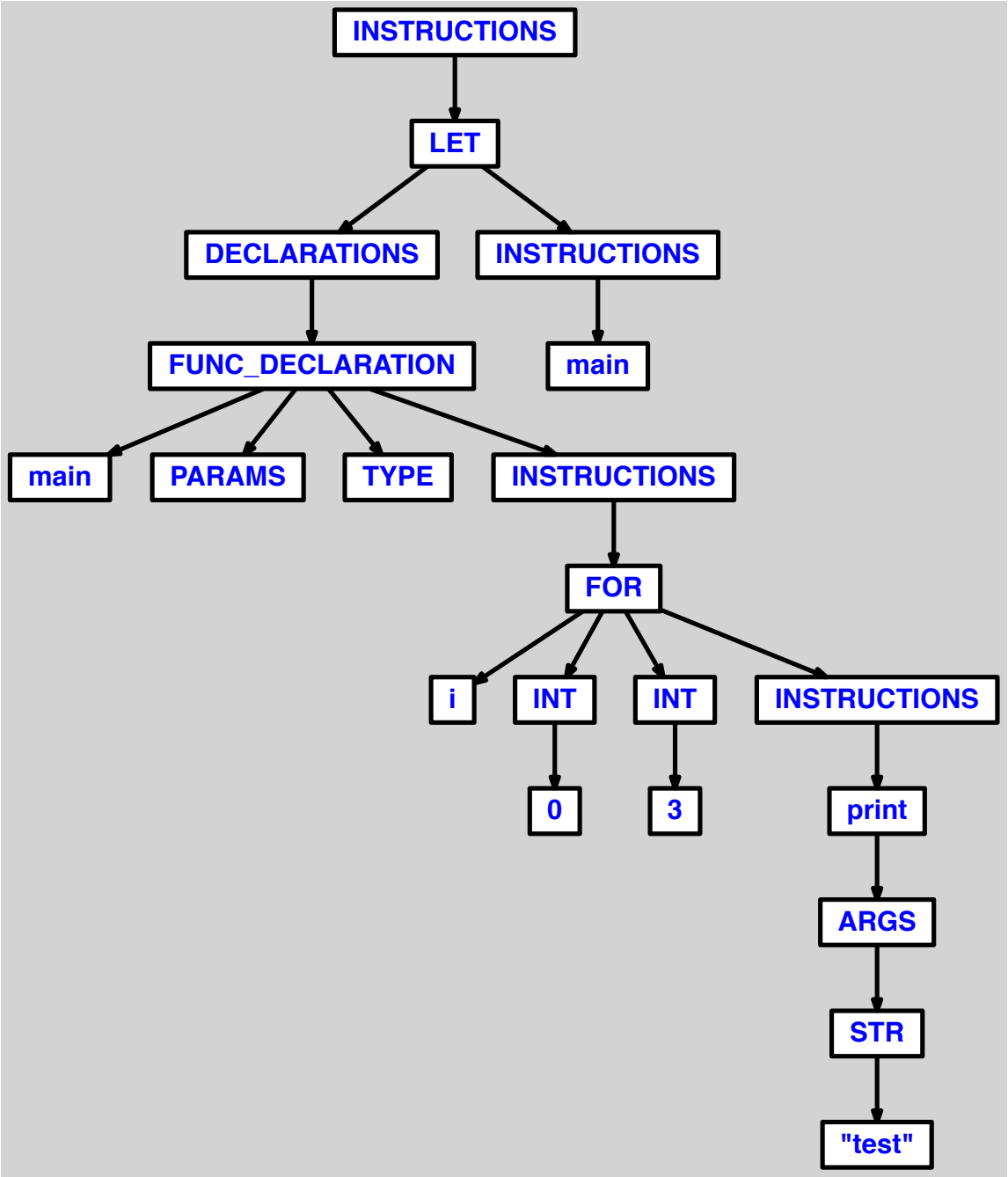
7.1.12 <for> avec oubli du < :>

```
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 < :=>

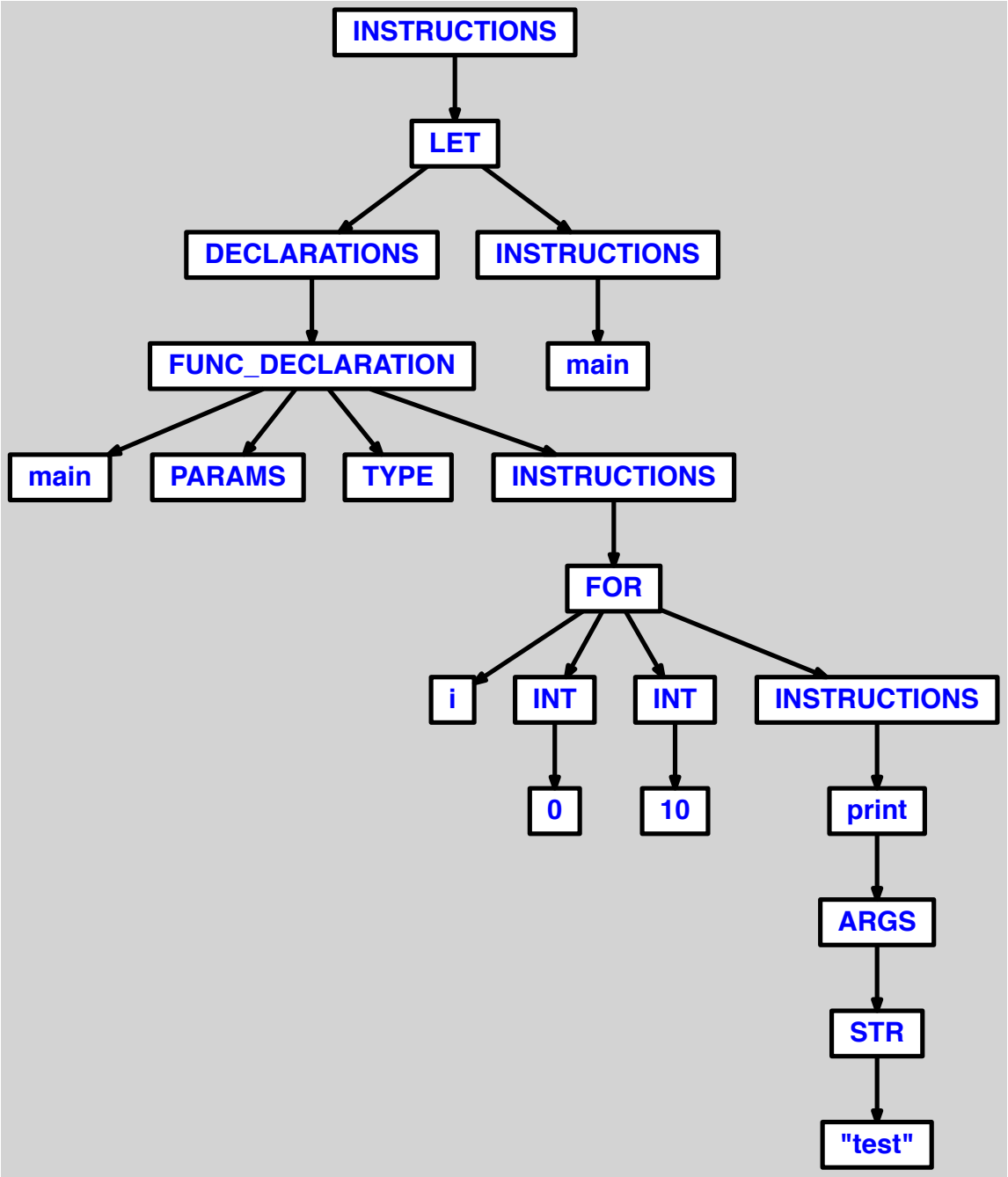
```
let
  function main() =
    for i 0 to 3 do
      print("test")
in main() end
```


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

7.2 OK

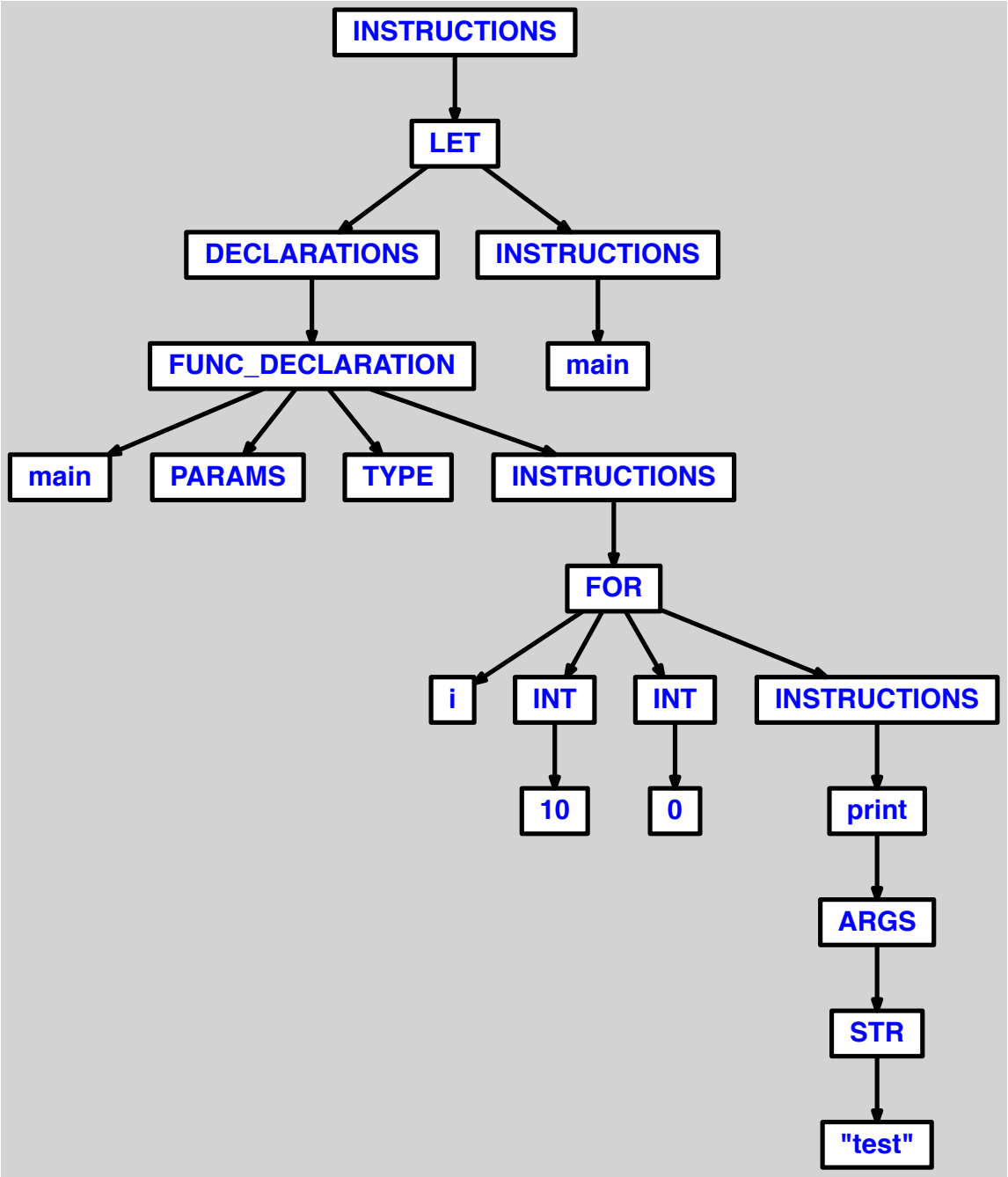
7.2.1 <for> simple croissant

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


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

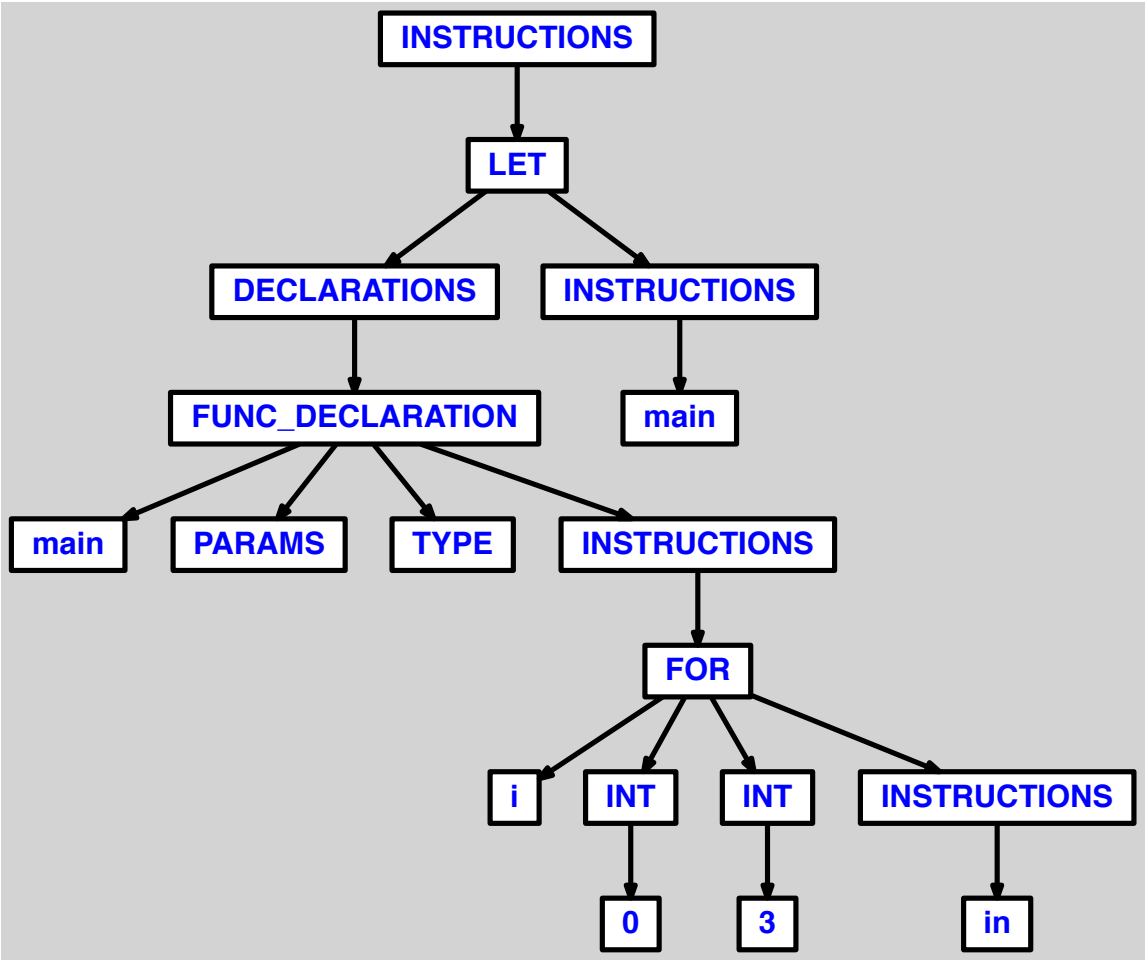
7.2.2 <for> simple decroissant

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


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

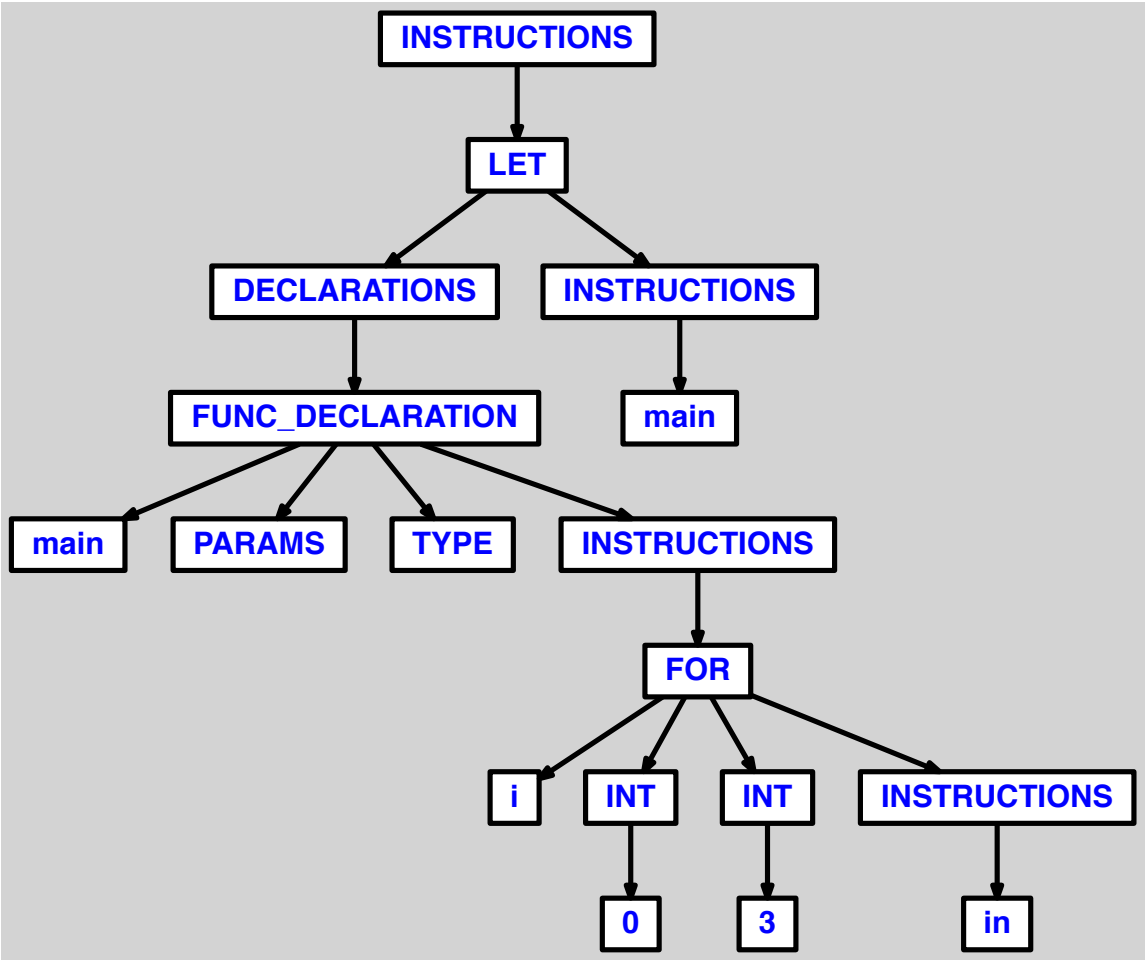
7.2.3 <for> sans instruction

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


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

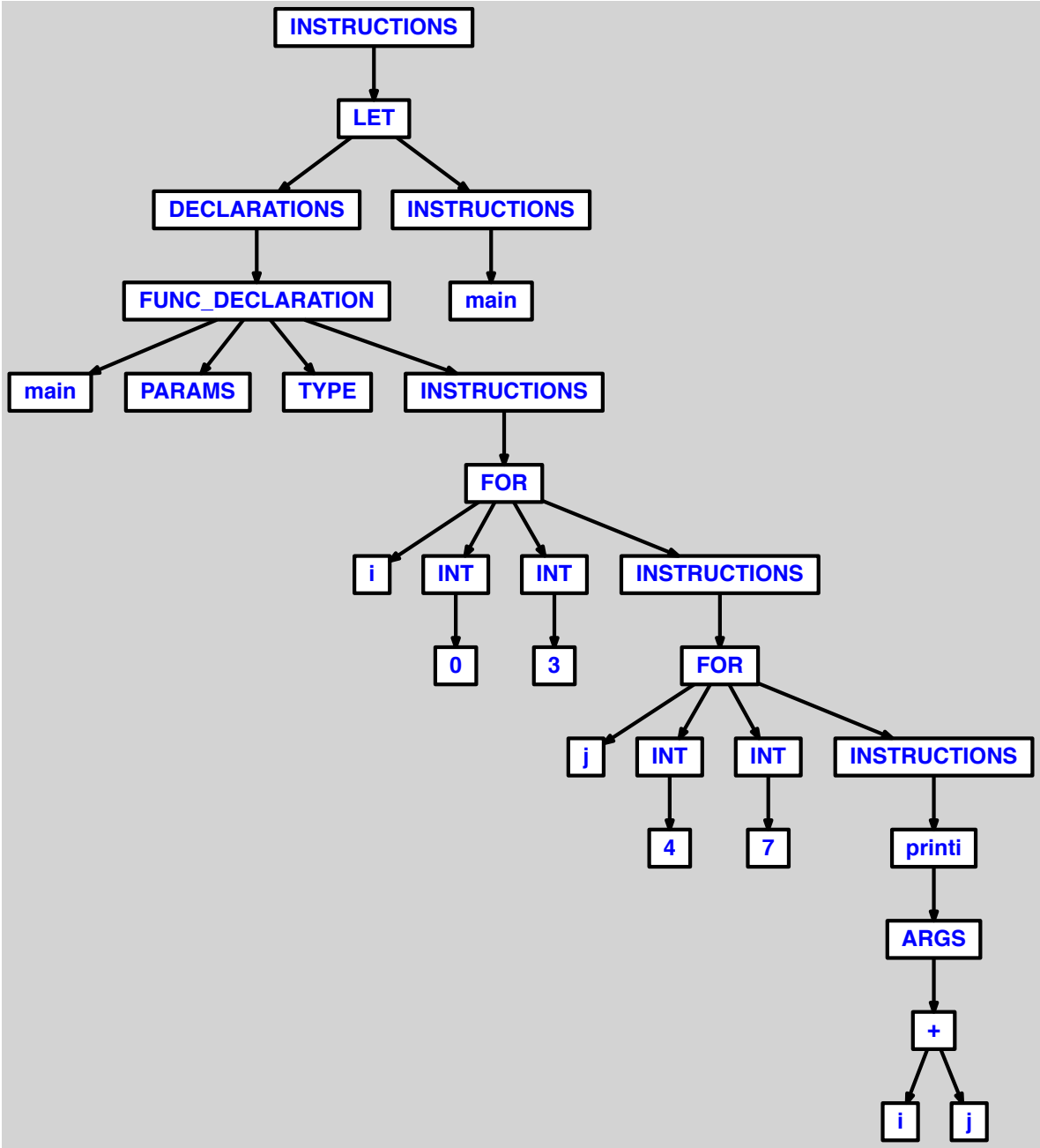
7.2.4 <for> avec ligne vide

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

in main() end
```


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

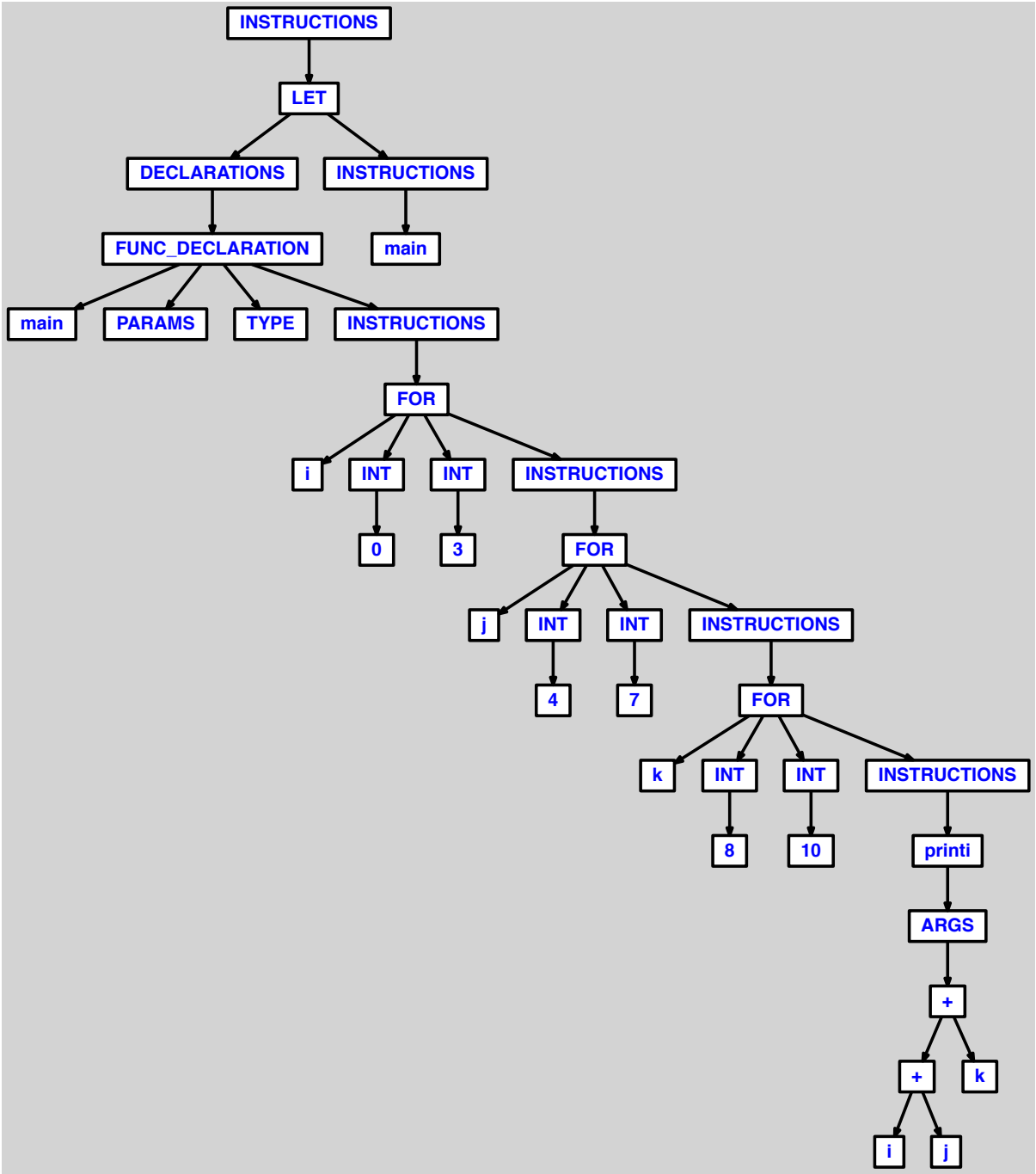
7.2.5 double-imbrication de <for>

```
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
main
main
PARAMS
TYPE
INSTRUCTIONS
FOR
i
INT
INT
INSTRUCTIONS
0
3
FOR
j
INT
INT
INSTRUCTIONS
4
7
printi
ARGS
+
i
j

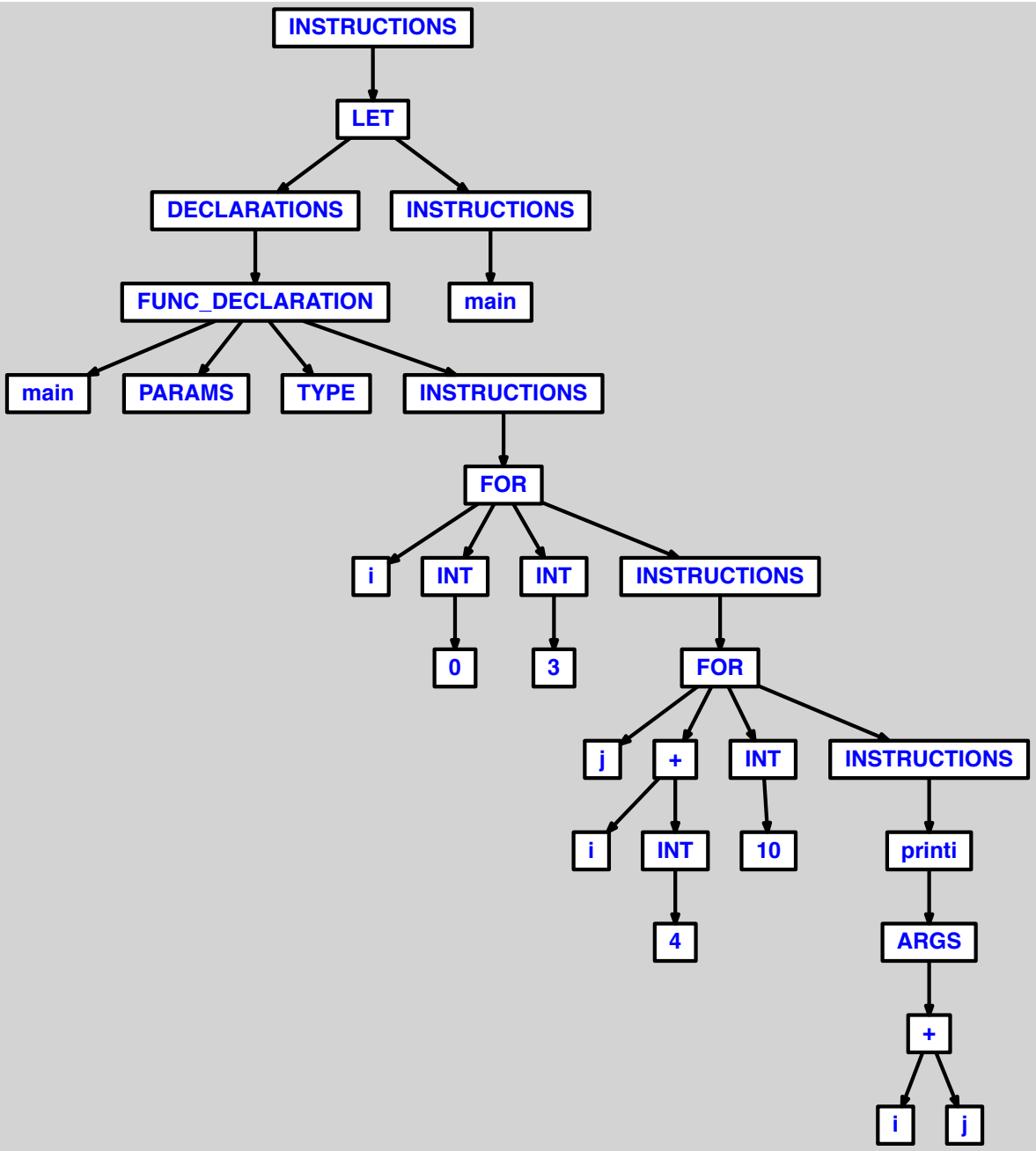
7.2.6 triple-imbrication de <for>

```
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
main
main
PARAMS
TYPE
INSTRUCTIONS
FOR
i
INT
INT
INSTRUCTIONS
0
3
FOR
j
INT
INT
INSTRUCTIONS
4
7
FOR
k
INT
INT
INSTRUCTIONS
8
10
printi
ARGS
+
+
k
i
j

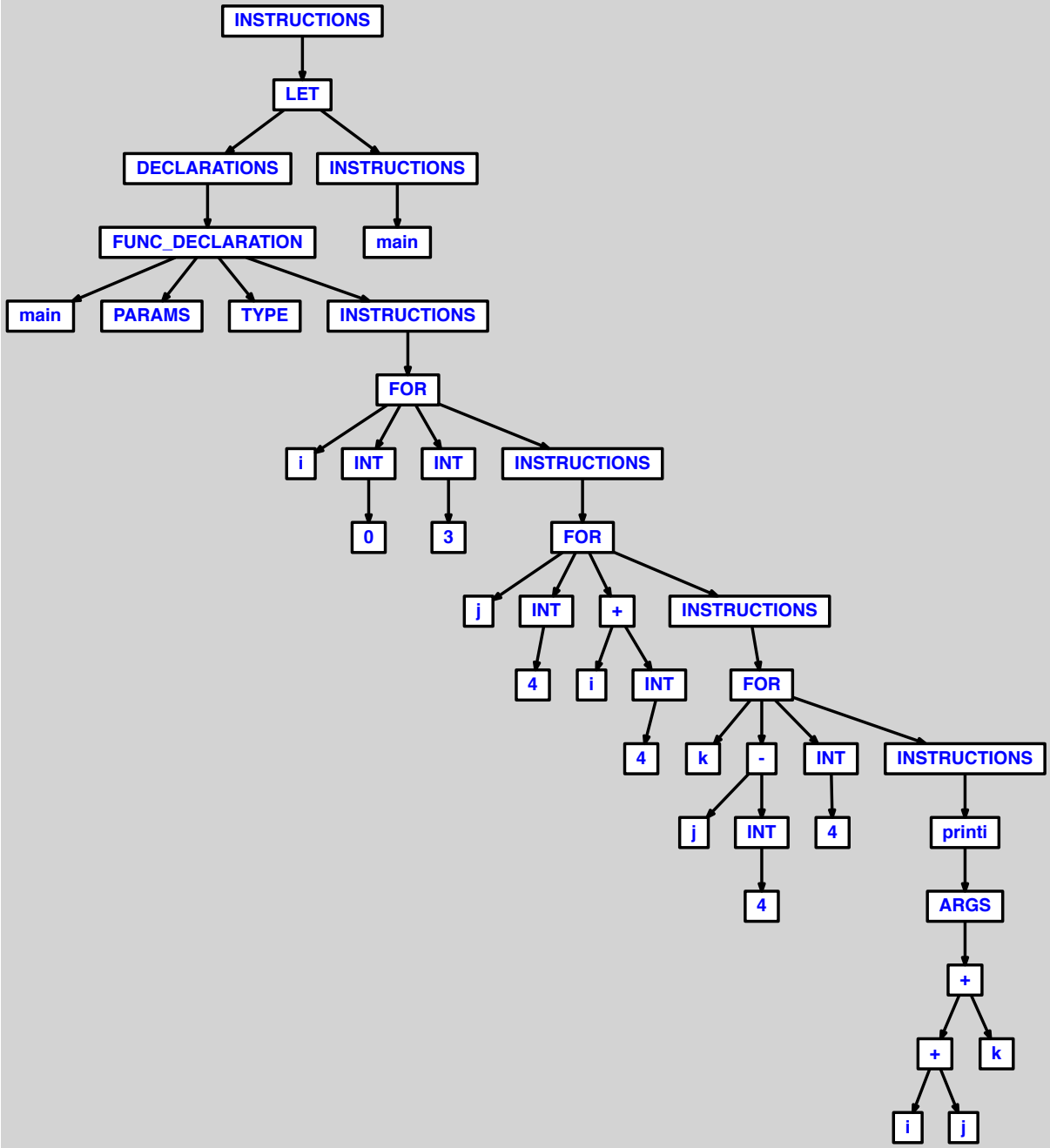
7.2.7 <for> avec reutilisation de compteur

```
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
main
main
PARAMS
TYPE
INSTRUCTIONS
FOR
i
INT
INT
INSTRUCTIONS
0
3
FOR
j
+
INT
INSTRUCTIONS
i
INT
10
printi
4
ARGS
+
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
```

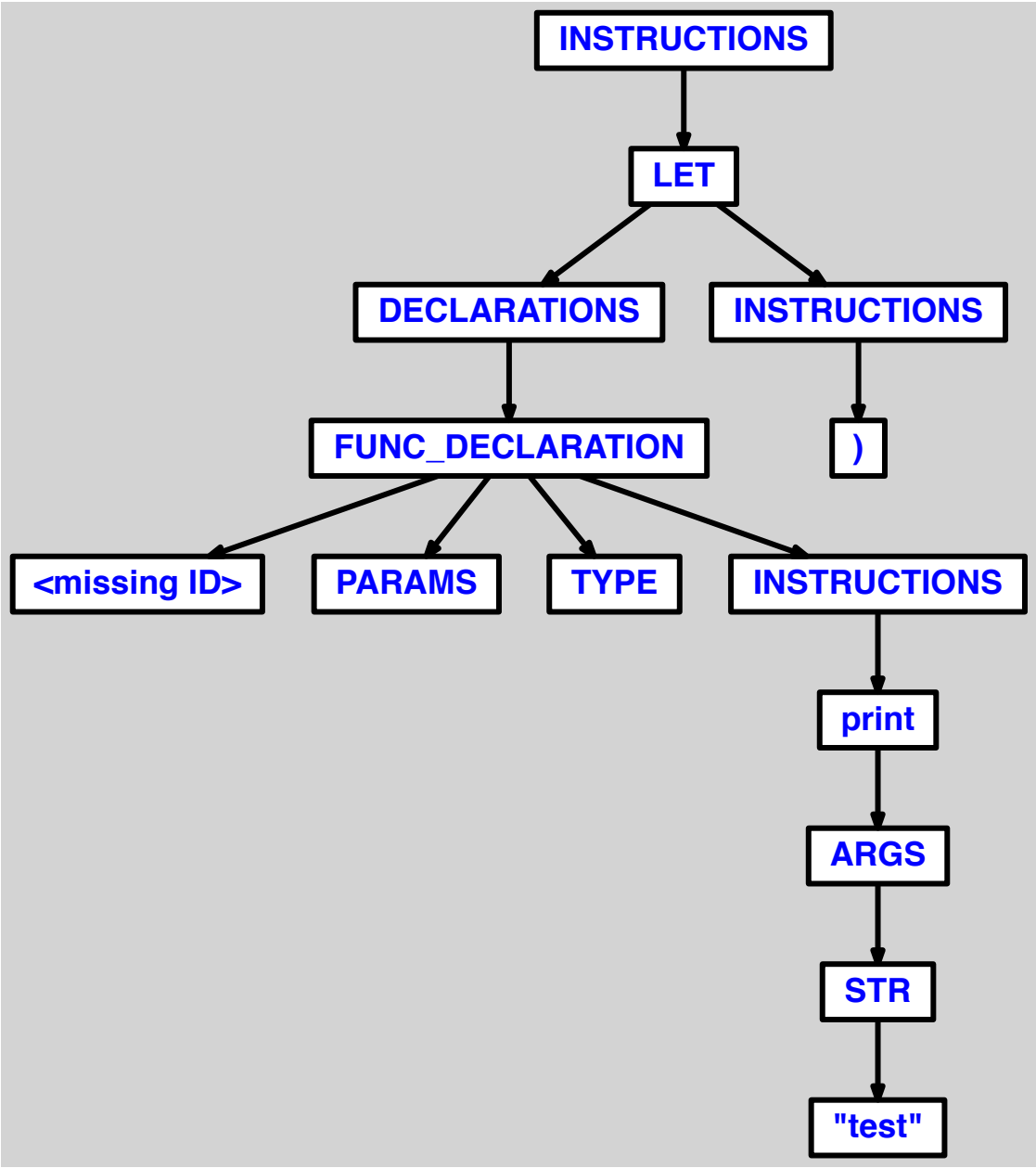

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
FOR
i
INT
INT
INSTRUCTIONS
0
3
FOR
j
INT
+
INSTRUCTIONS
4
i
INT
FOR
4
k
-
INT
INSTRUCTIONS
j
INT
4
printi
4
ARGS
+
+
k
i
j

8 function

8.1 KO

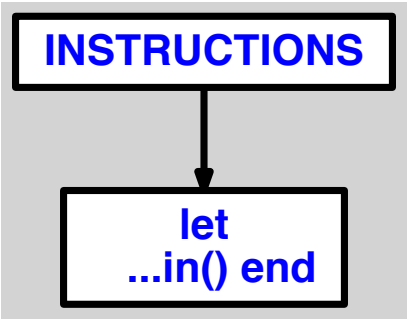
8.1.1 fonction identifiee par caractere special

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


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

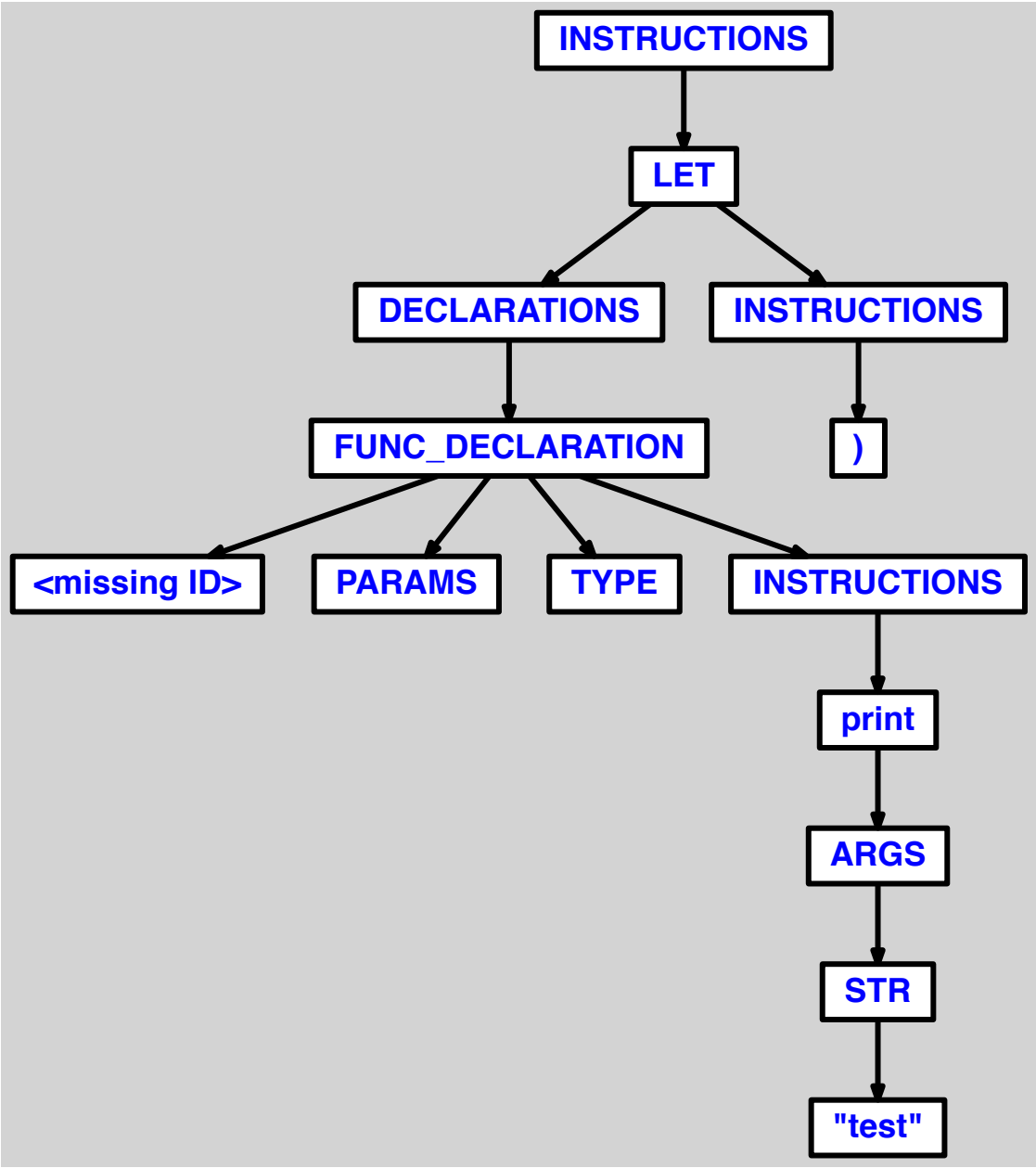
8.1.2 oubli de <function>

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


INSTRUCTIONS
let
...in() end

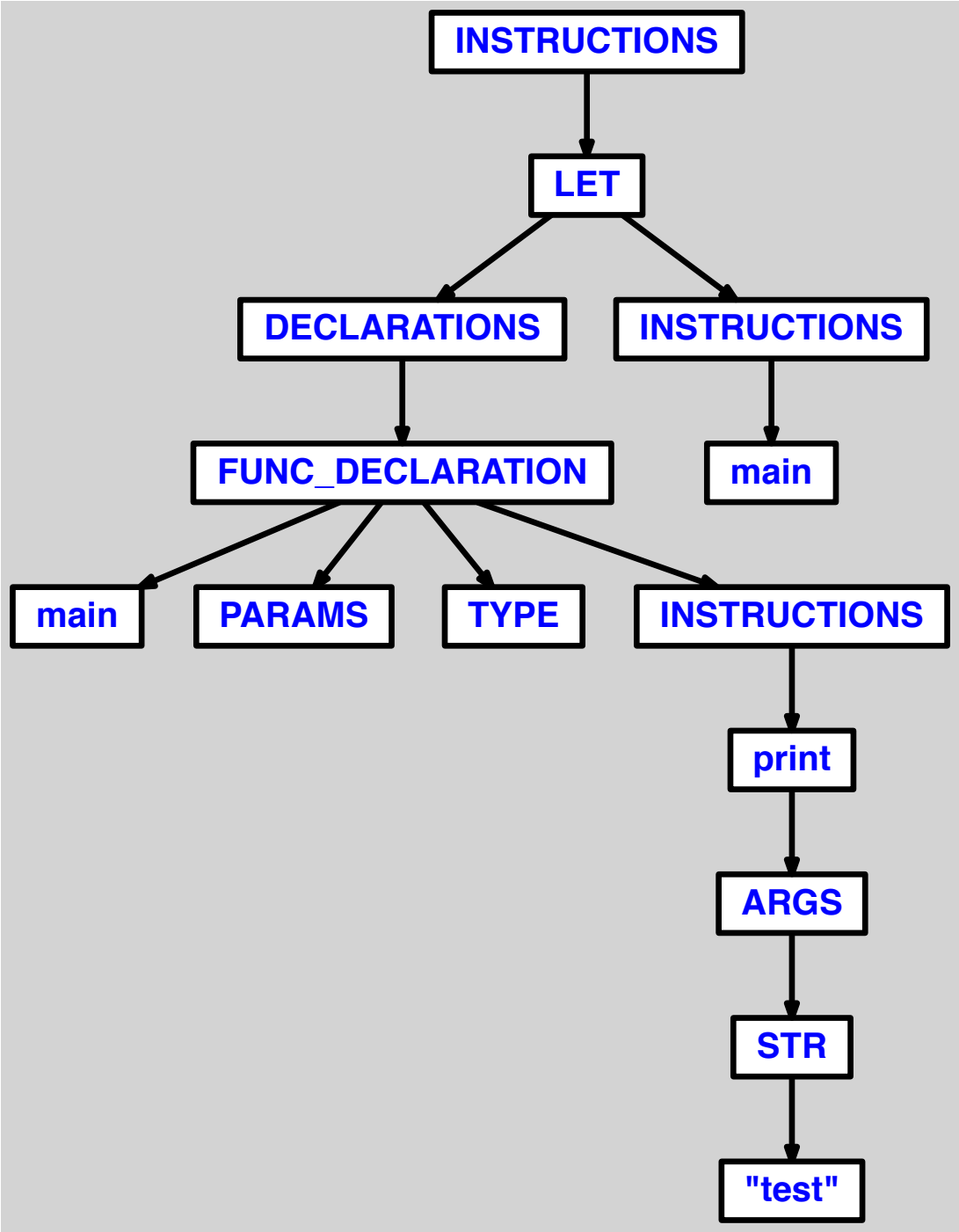
8.1.3 fonction sans identifiant

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


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

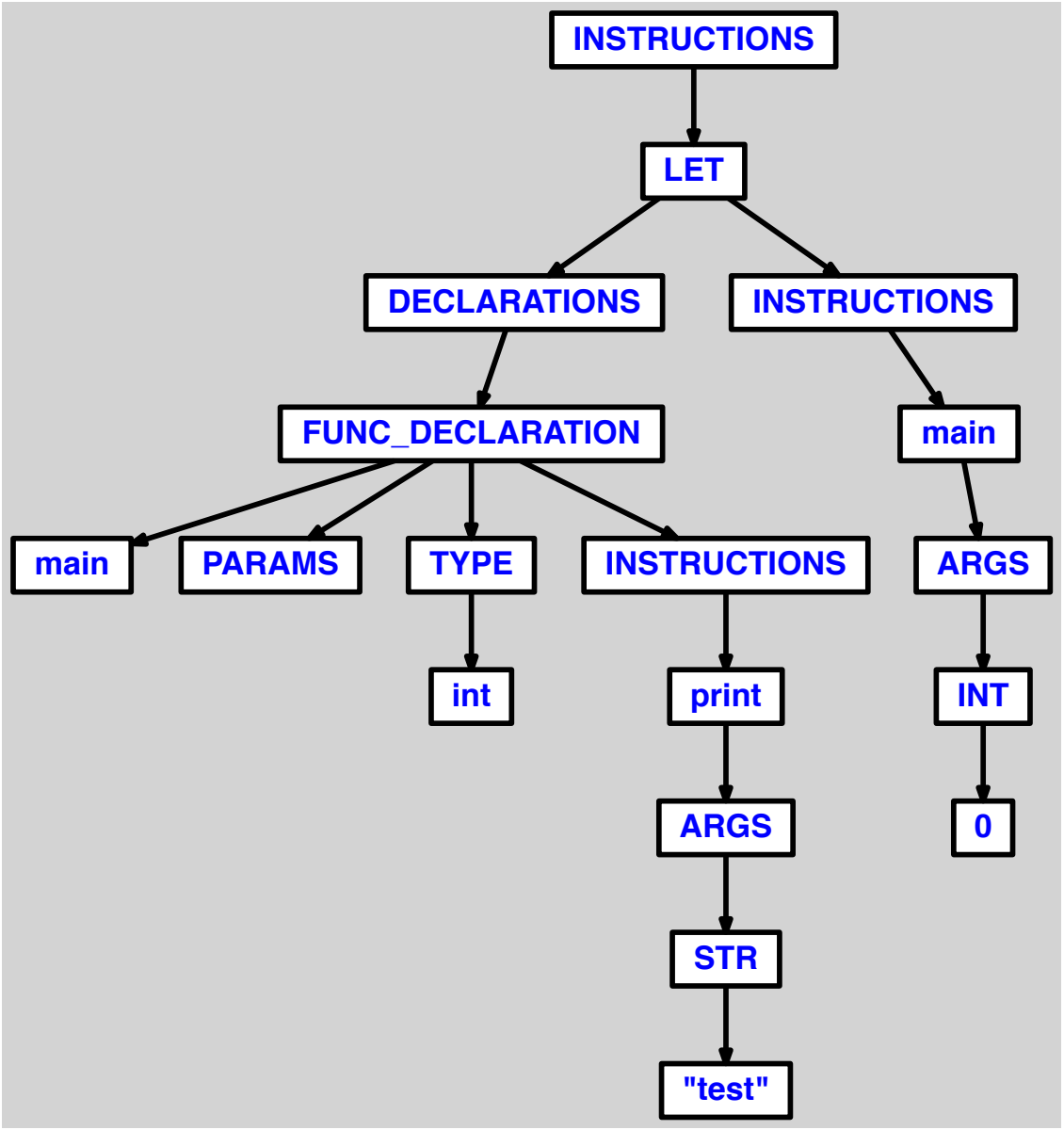
8.1.4 oubli de <(>

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


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

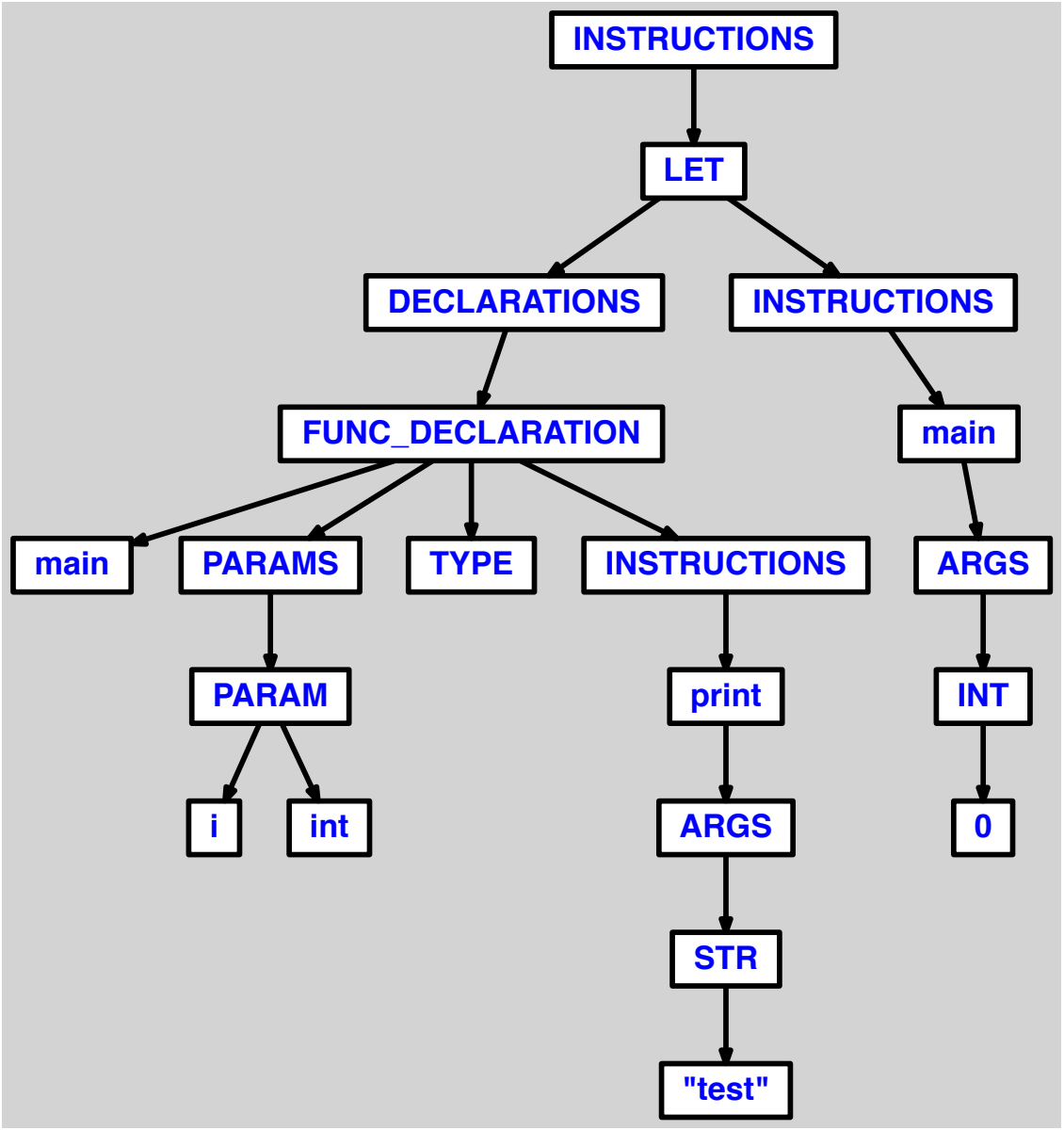
8.1.5 oubli d'identifiant de parametre

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


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

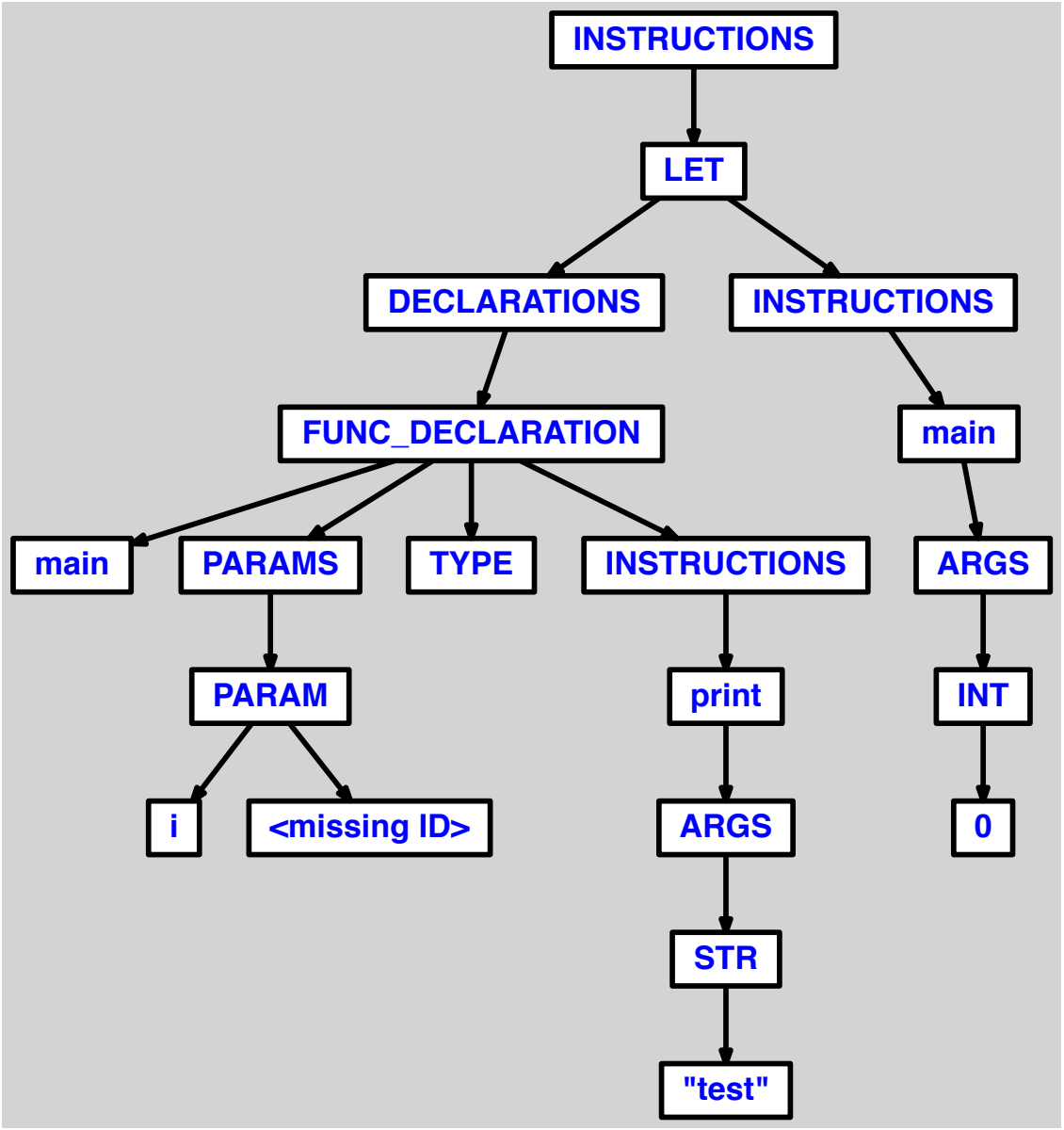
8.1.6 oubli de < :> pour parametre

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


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

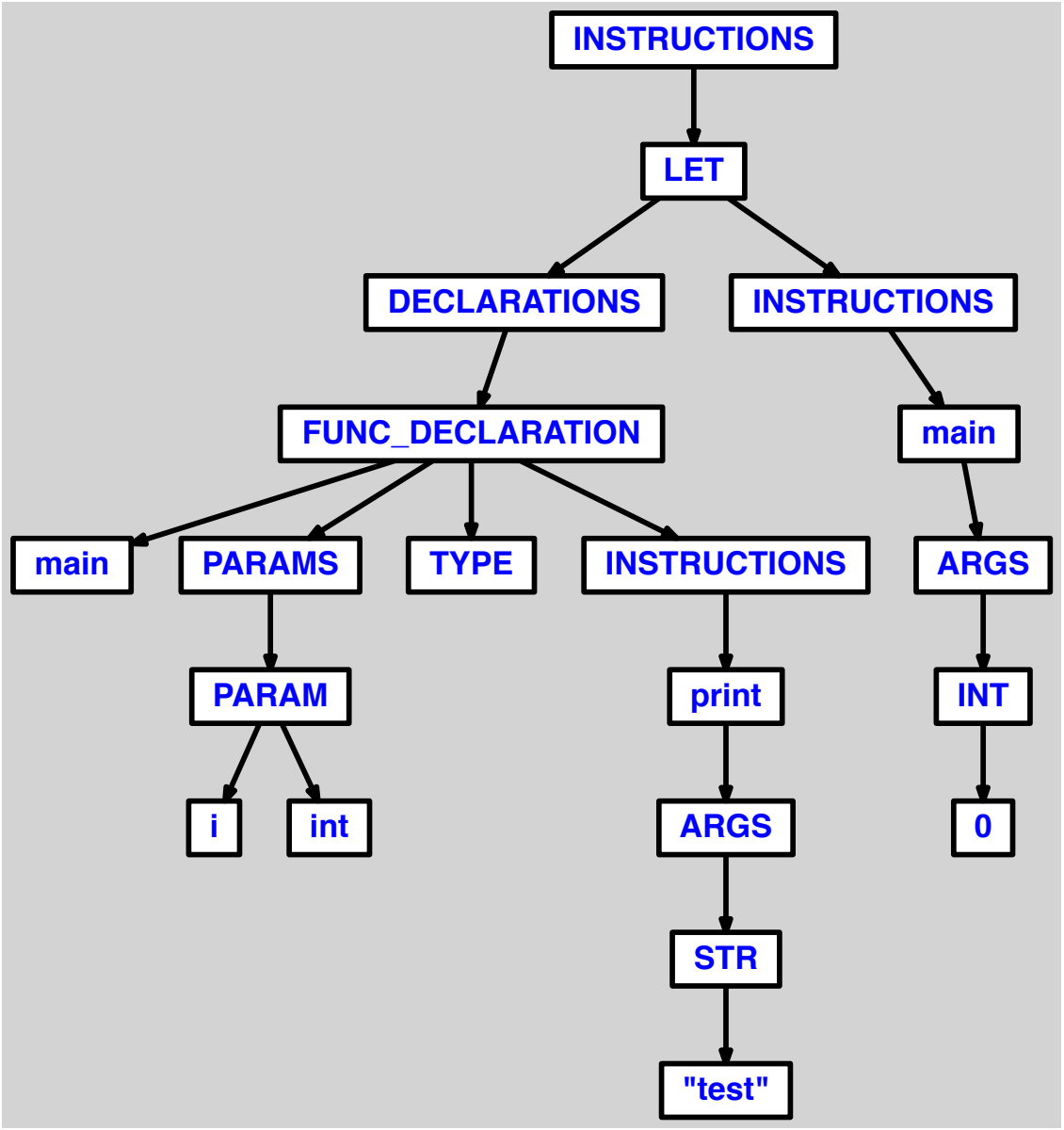
8.1.7 oubli de type de parametre

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


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

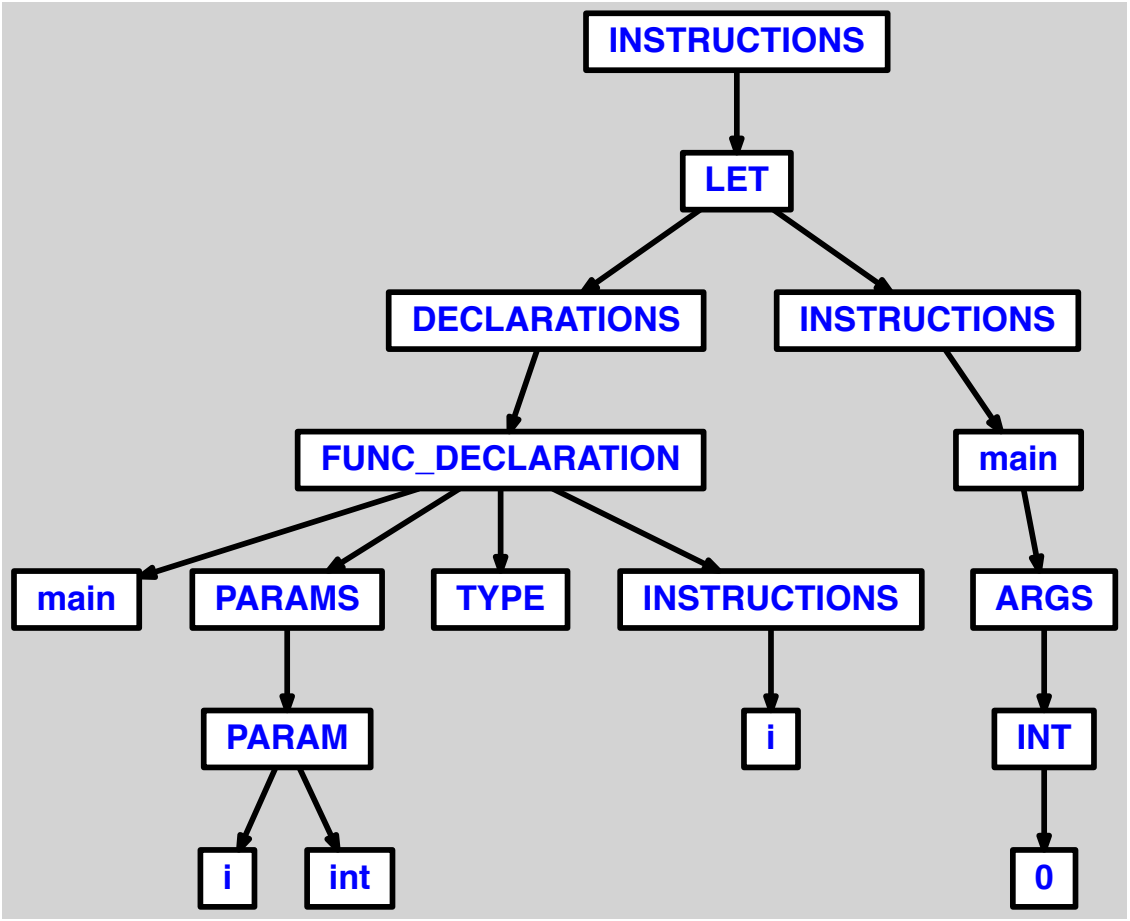
8.1.8 oubli de <)>

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


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

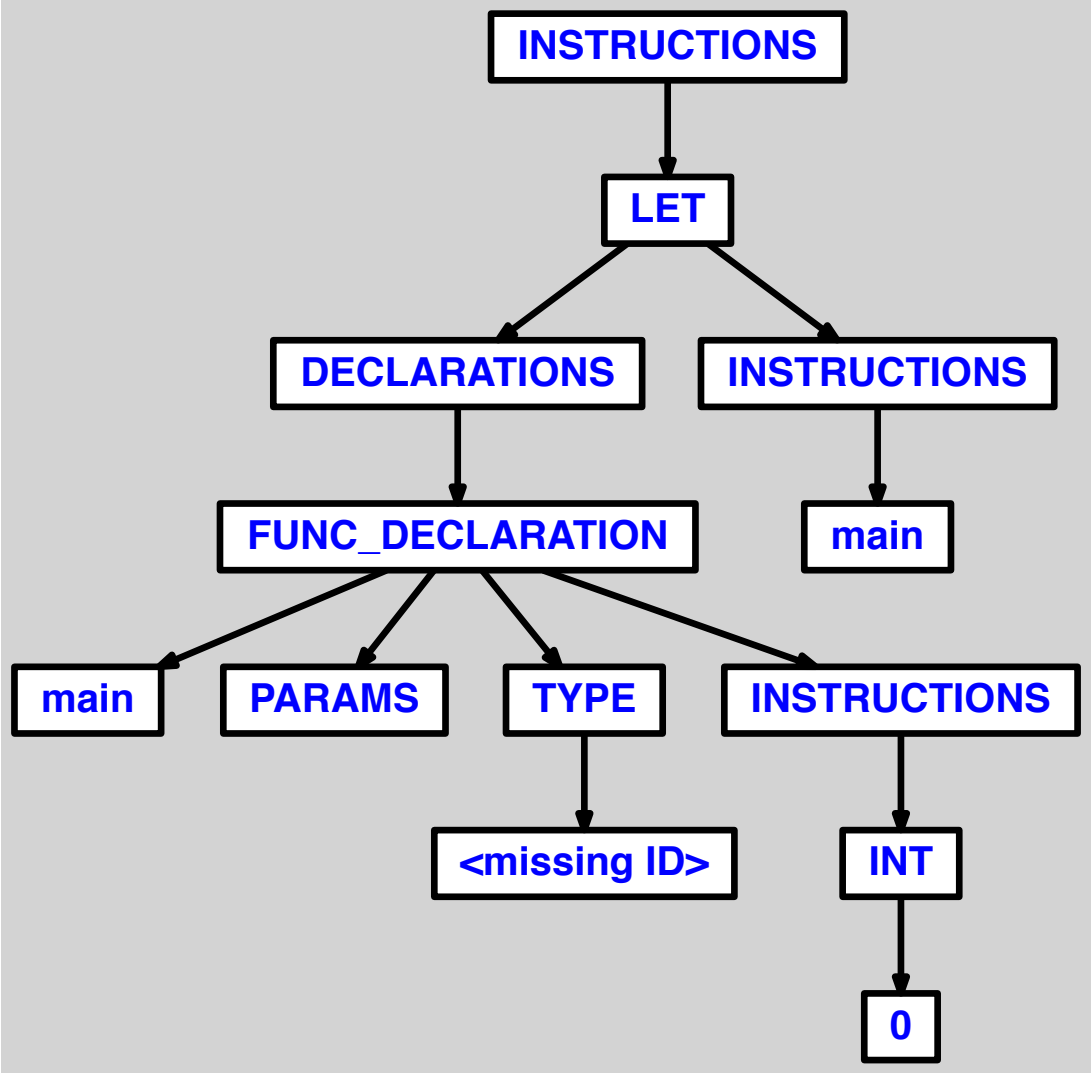
8.1.9 oubli de < :> pour type de fonction

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


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

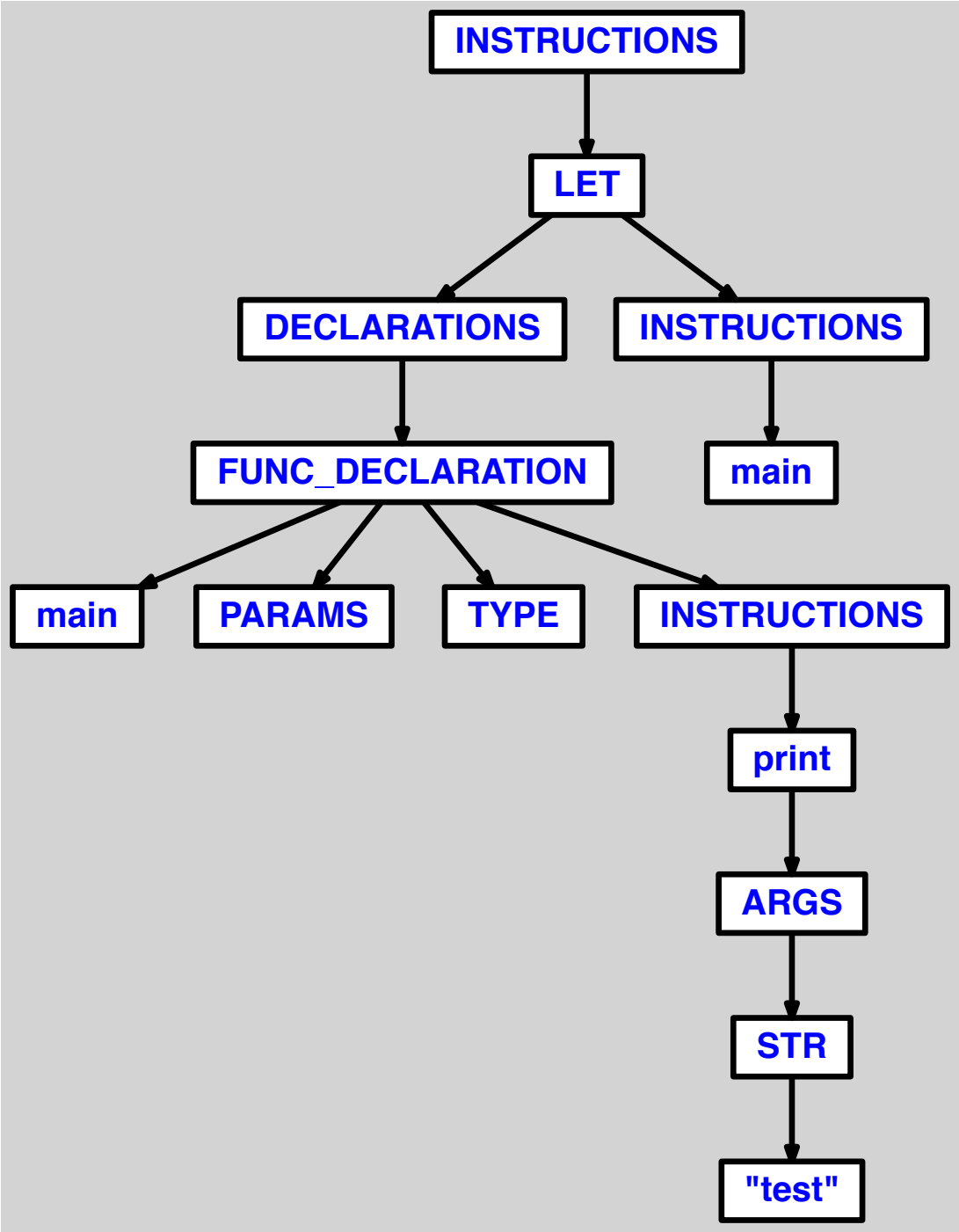
8.1.10 < :> pour type de fonction present mais pas de type

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


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

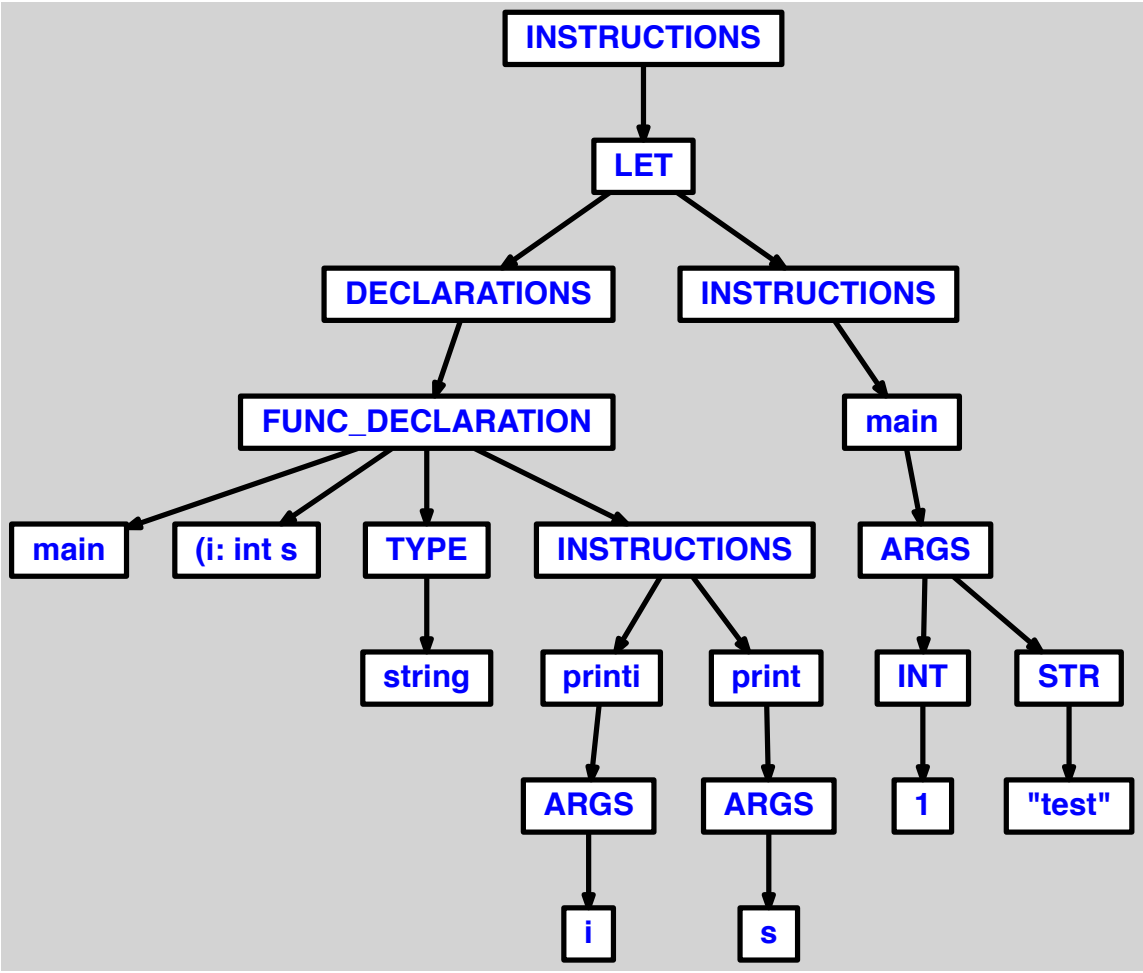
8.1.11 oubli de <=>

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


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

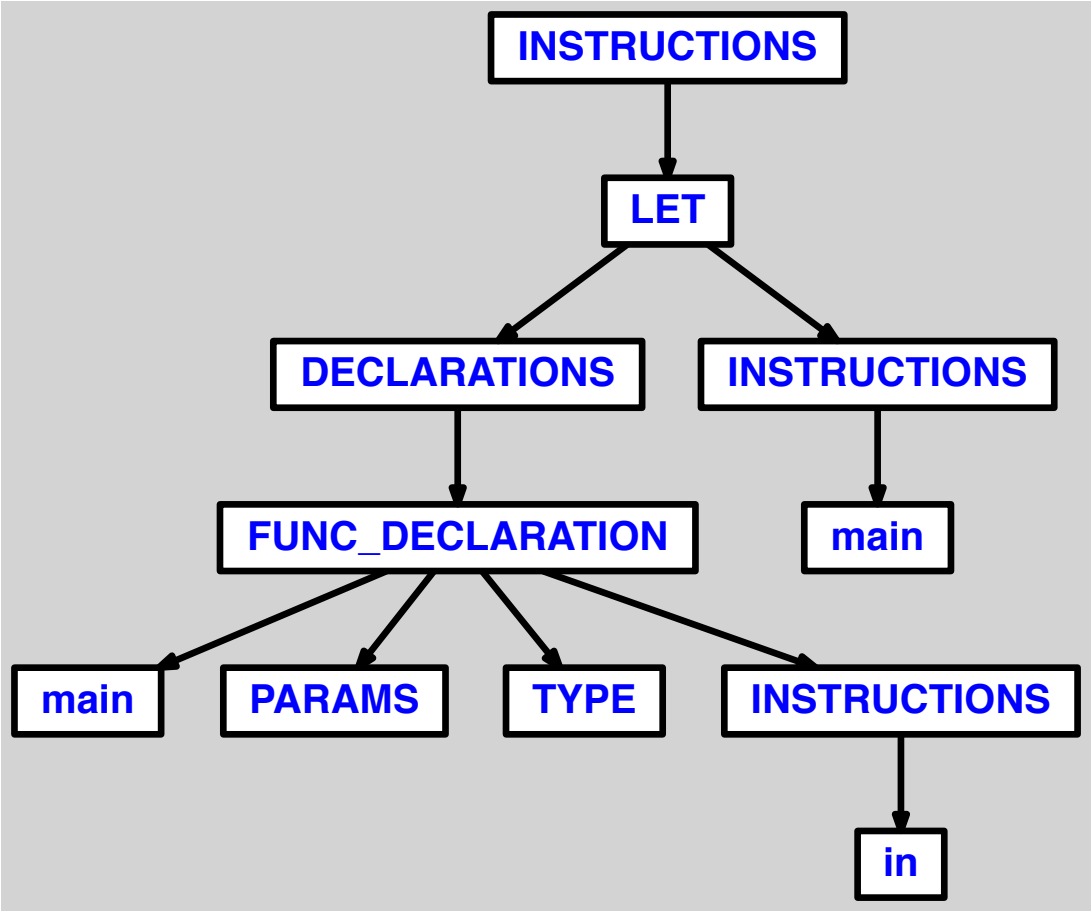
8.1.12 parametres mal separes

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


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

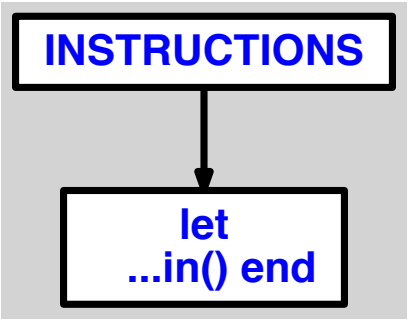
8.1.13 fonction sans instruction

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
in

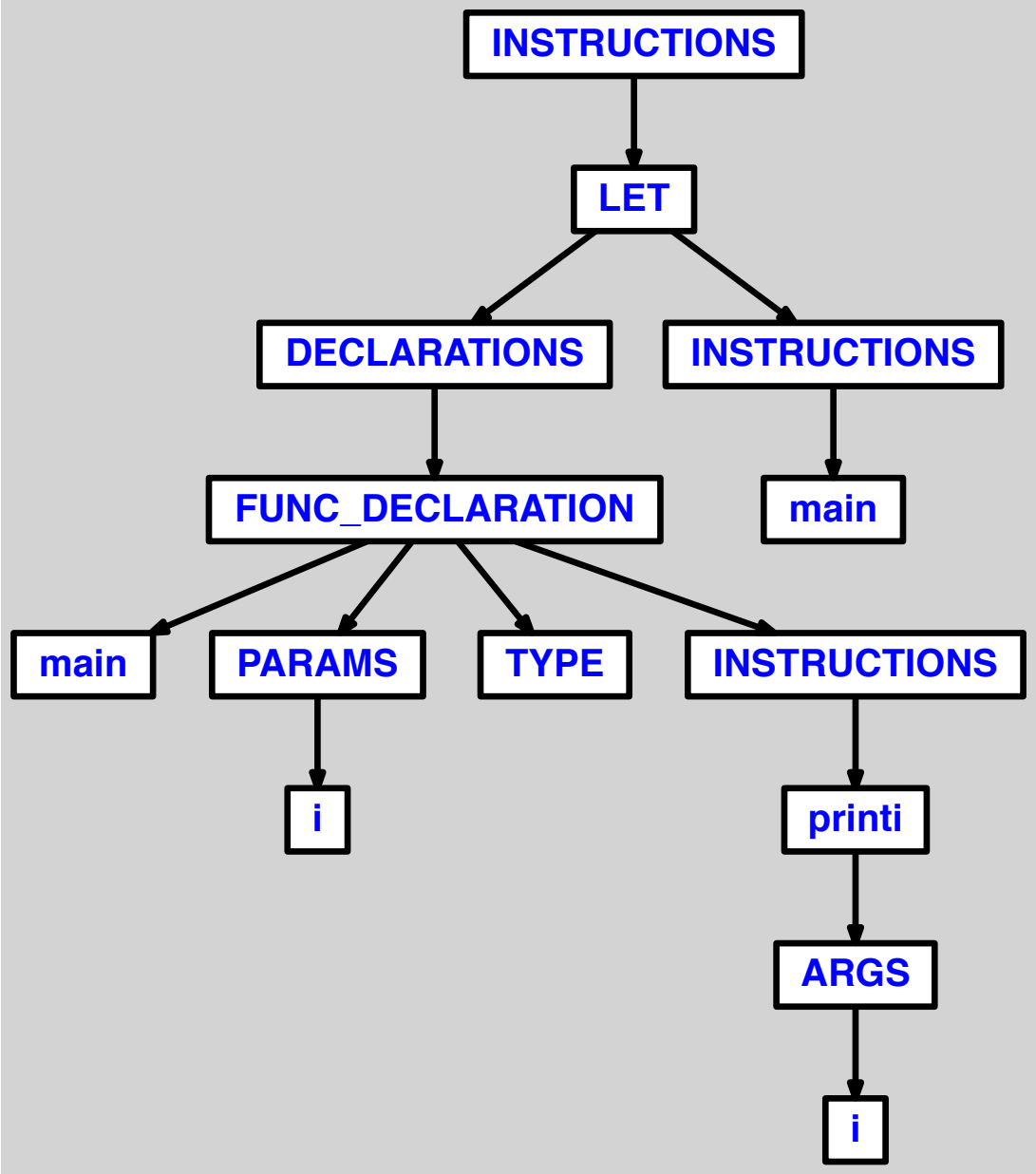
8.1.14 <function> mal ecrit

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


INSTRUCTIONS
let
...in() end

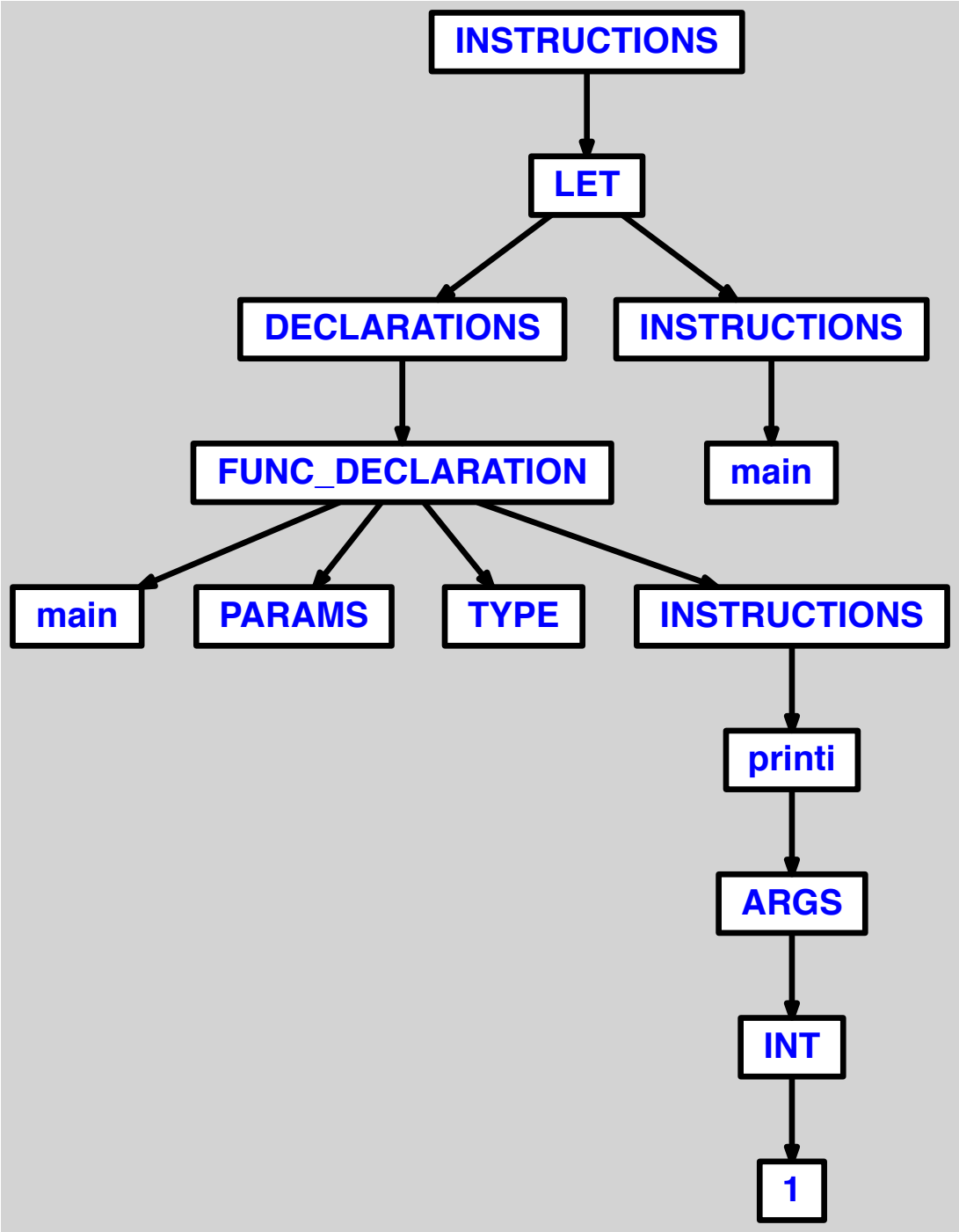
8.1.15 fonction avec identifiant seulement en parametre

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


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

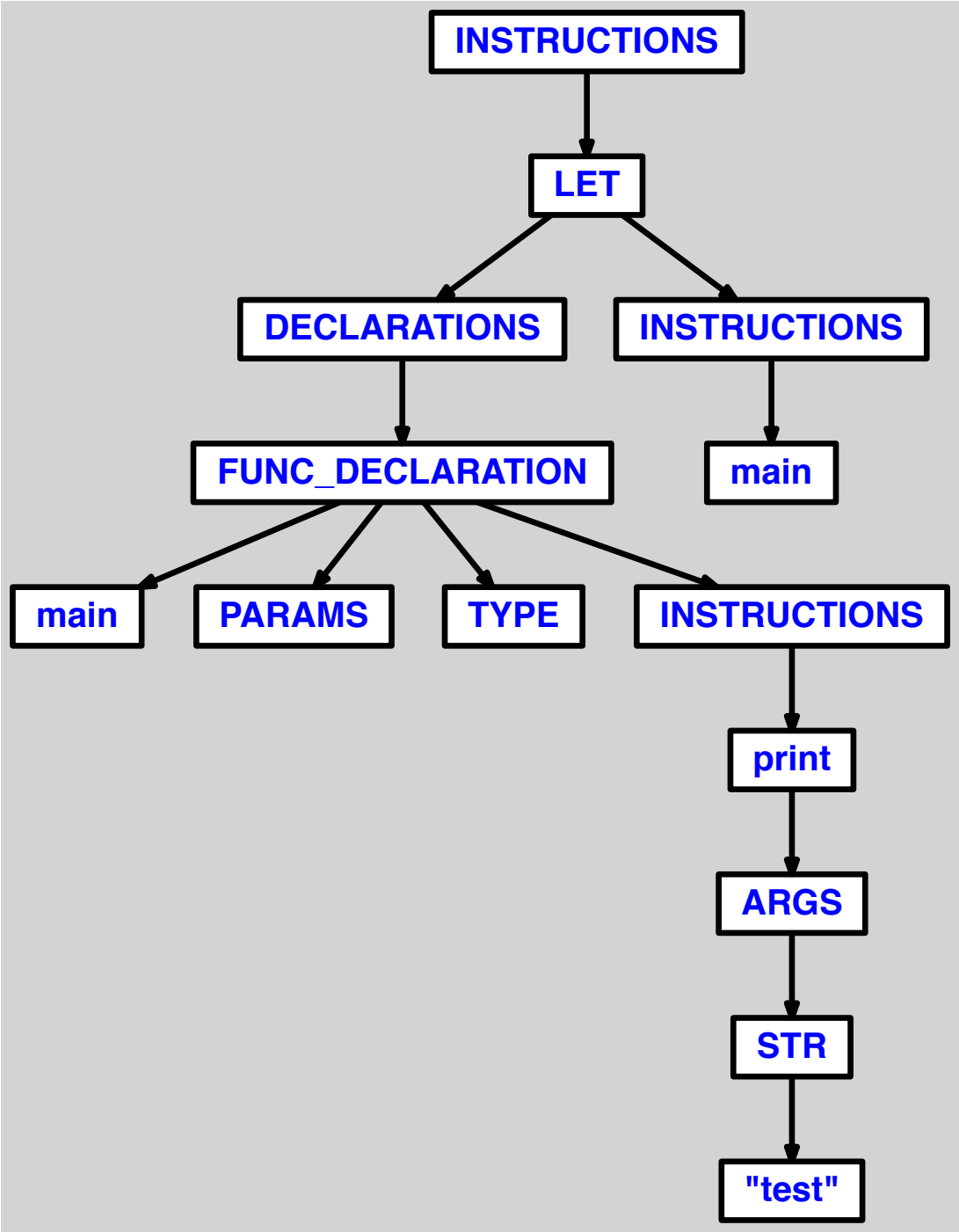
8.1.16 fonction avec entier directement en parametre

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


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

8.1.17 fonction avec chaine directement en parametre

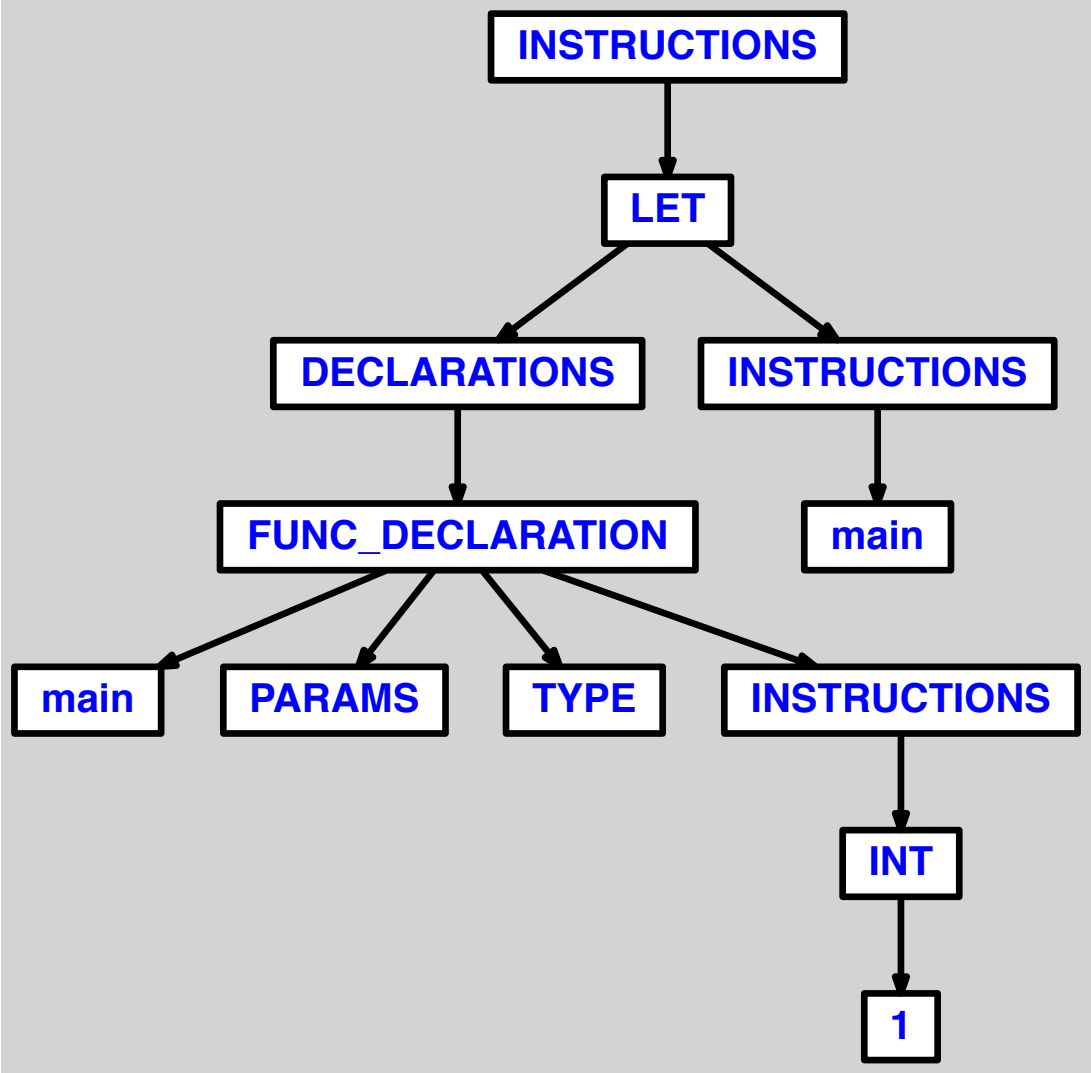
```
let
  function main("test") = print("test")
in main() end
```


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

8.2 OK

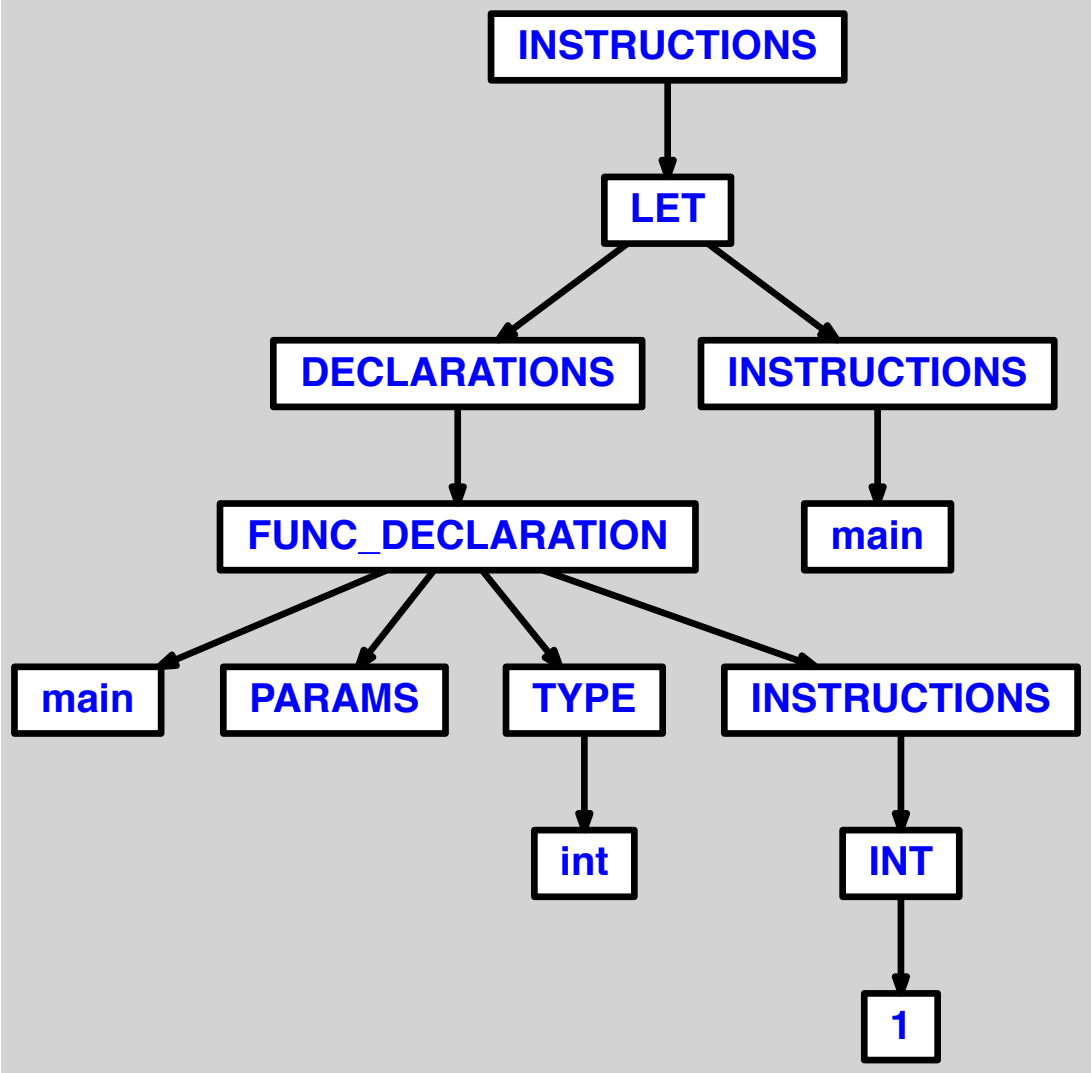
8.2.1 fonction definissant un entier

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
INT
1

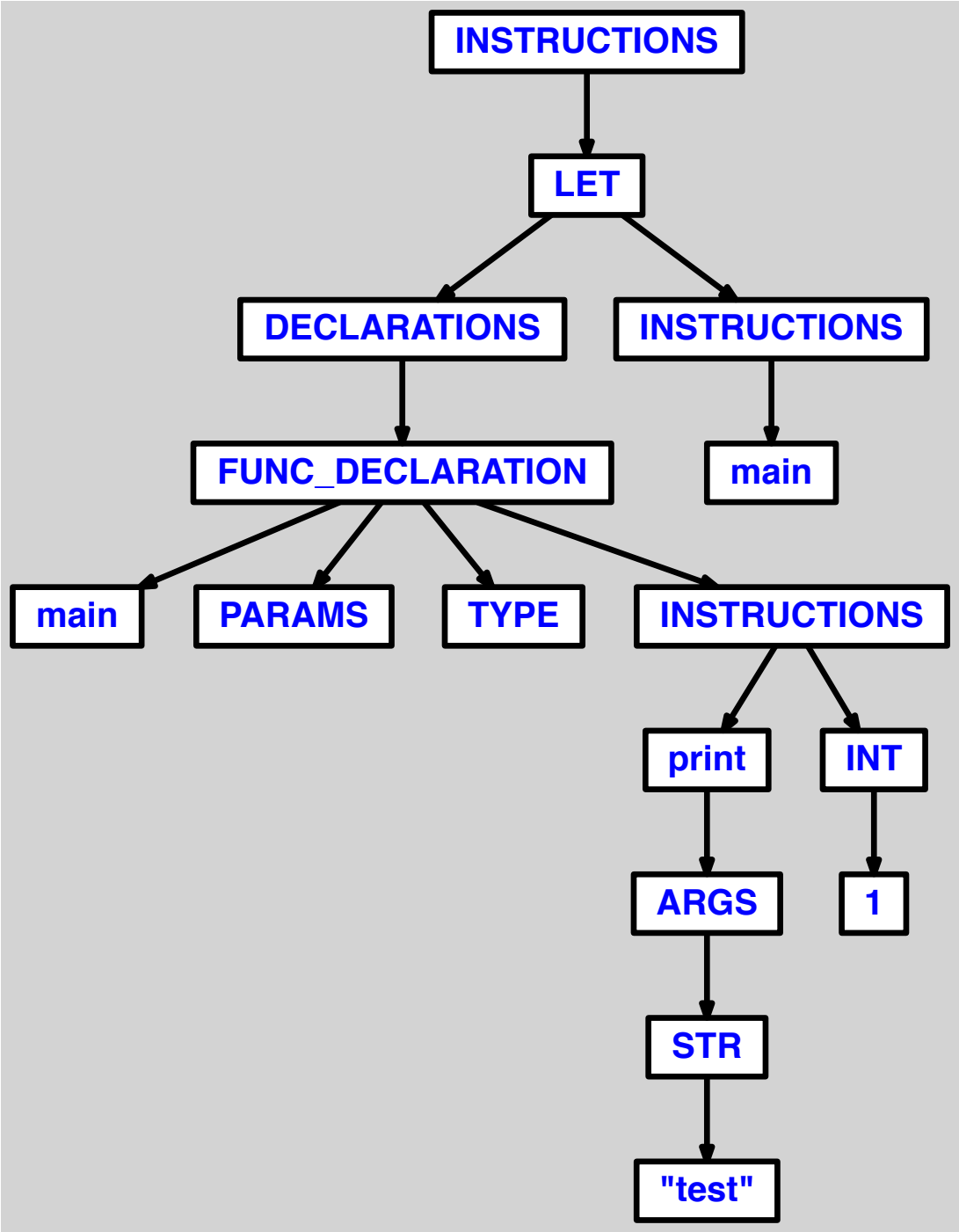
8.2.2 fonction simple, avec declaration du type de retour <int> et instruction de retour

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
int
INT
1

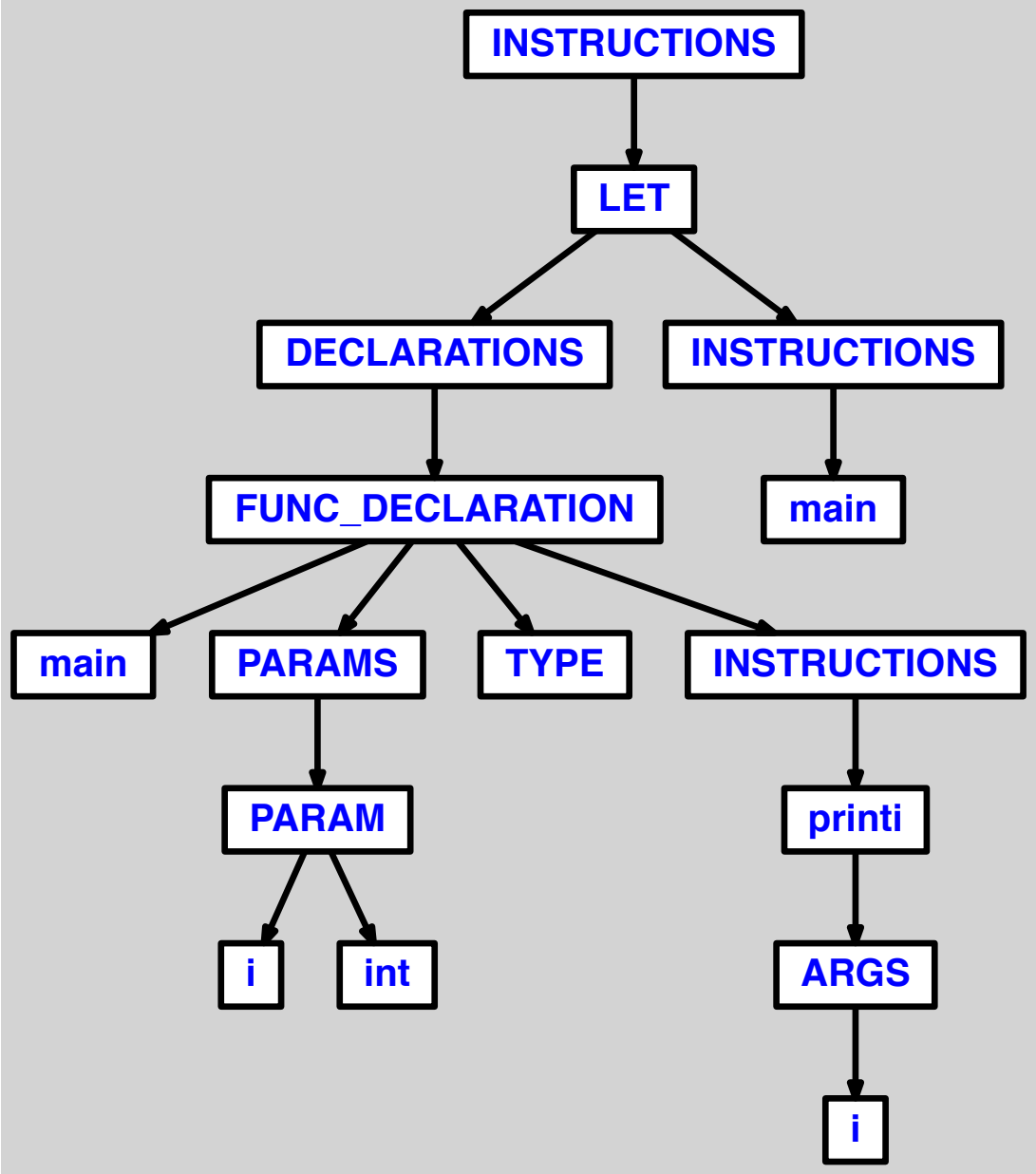
8.2.3 fonction simple, avec instruction simple et instruction retournant un entier

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


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

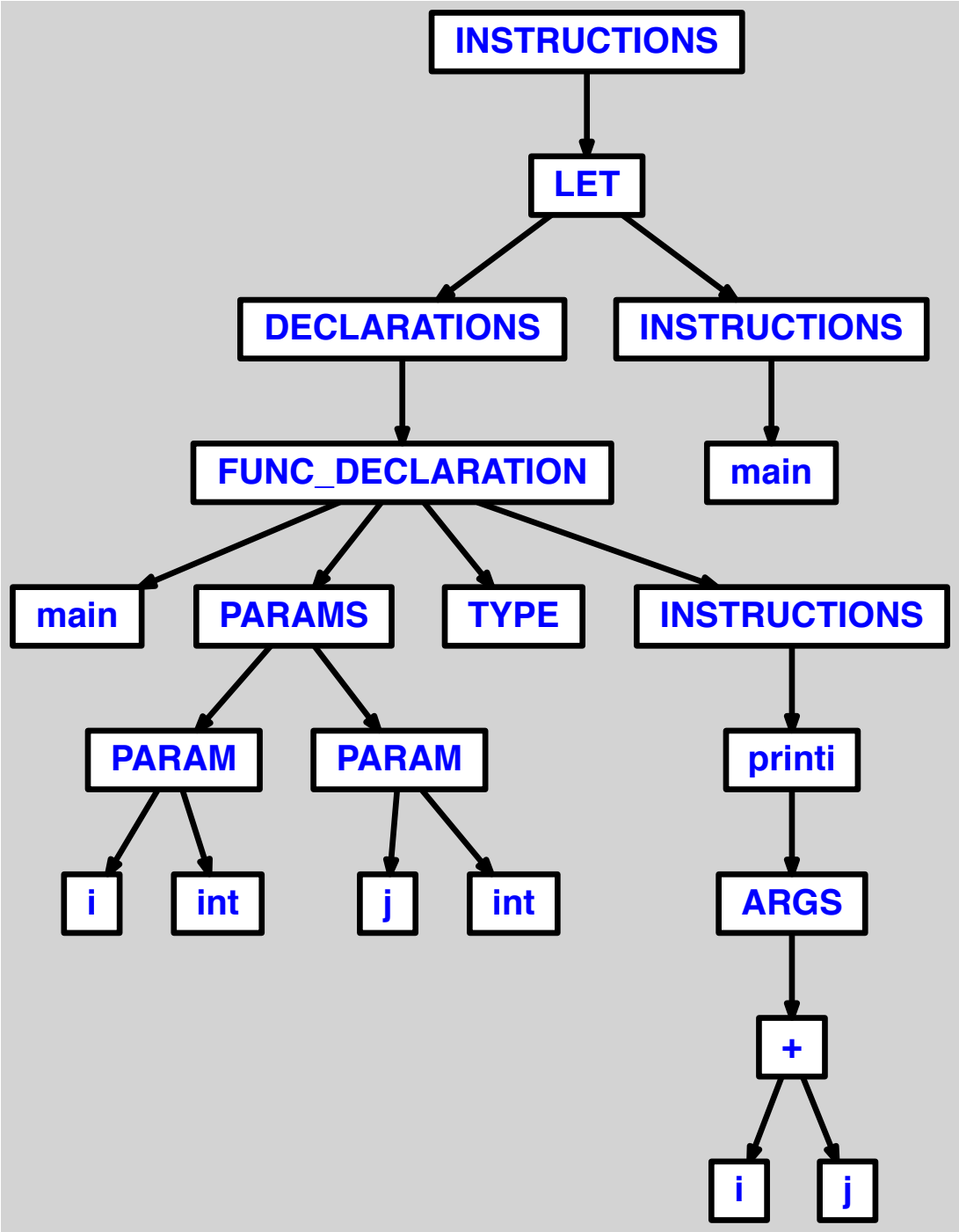
8.2.4 fonction avec 1 parametre entier

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


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

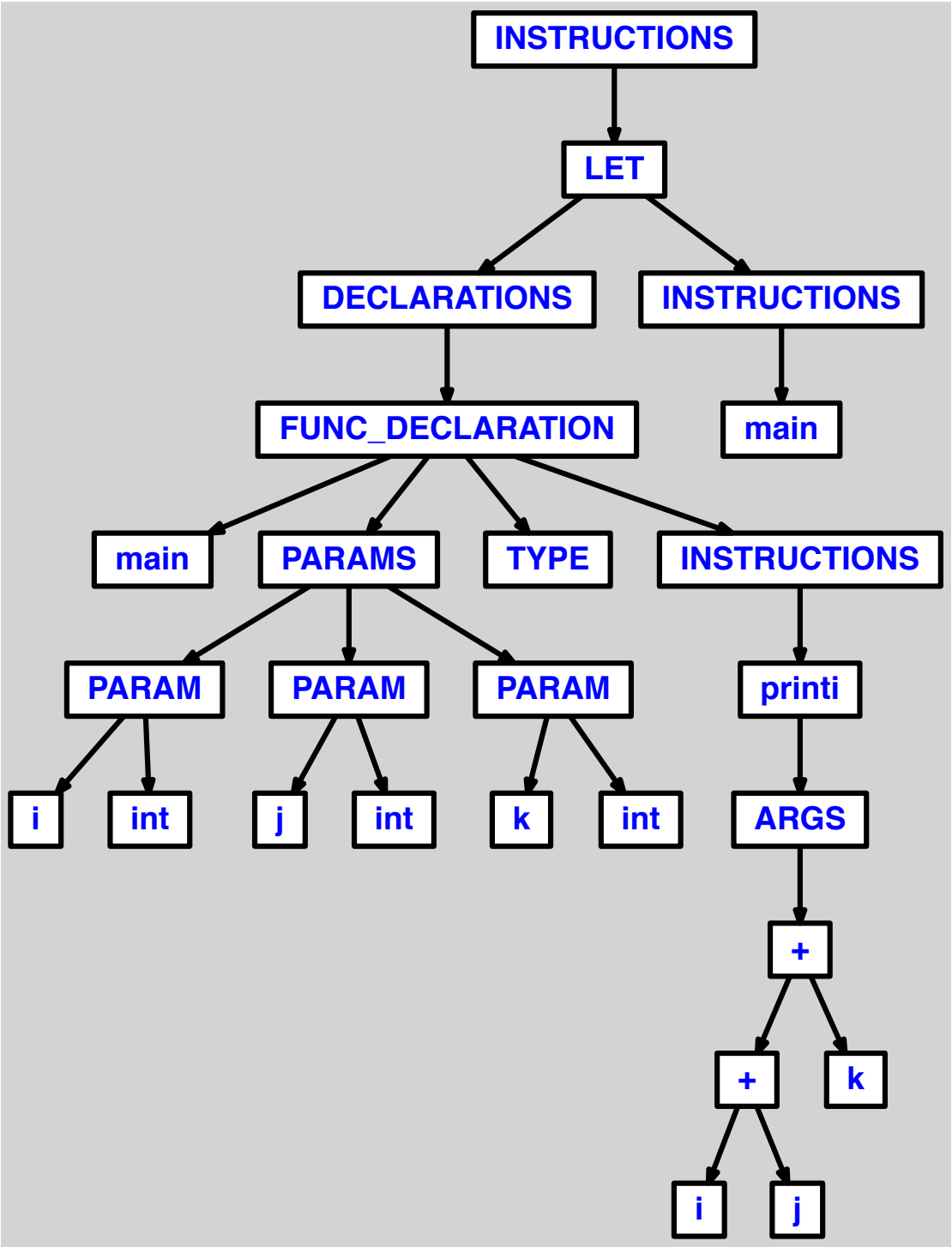
8.2.5 fonction avec 2 parametres entiers

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


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

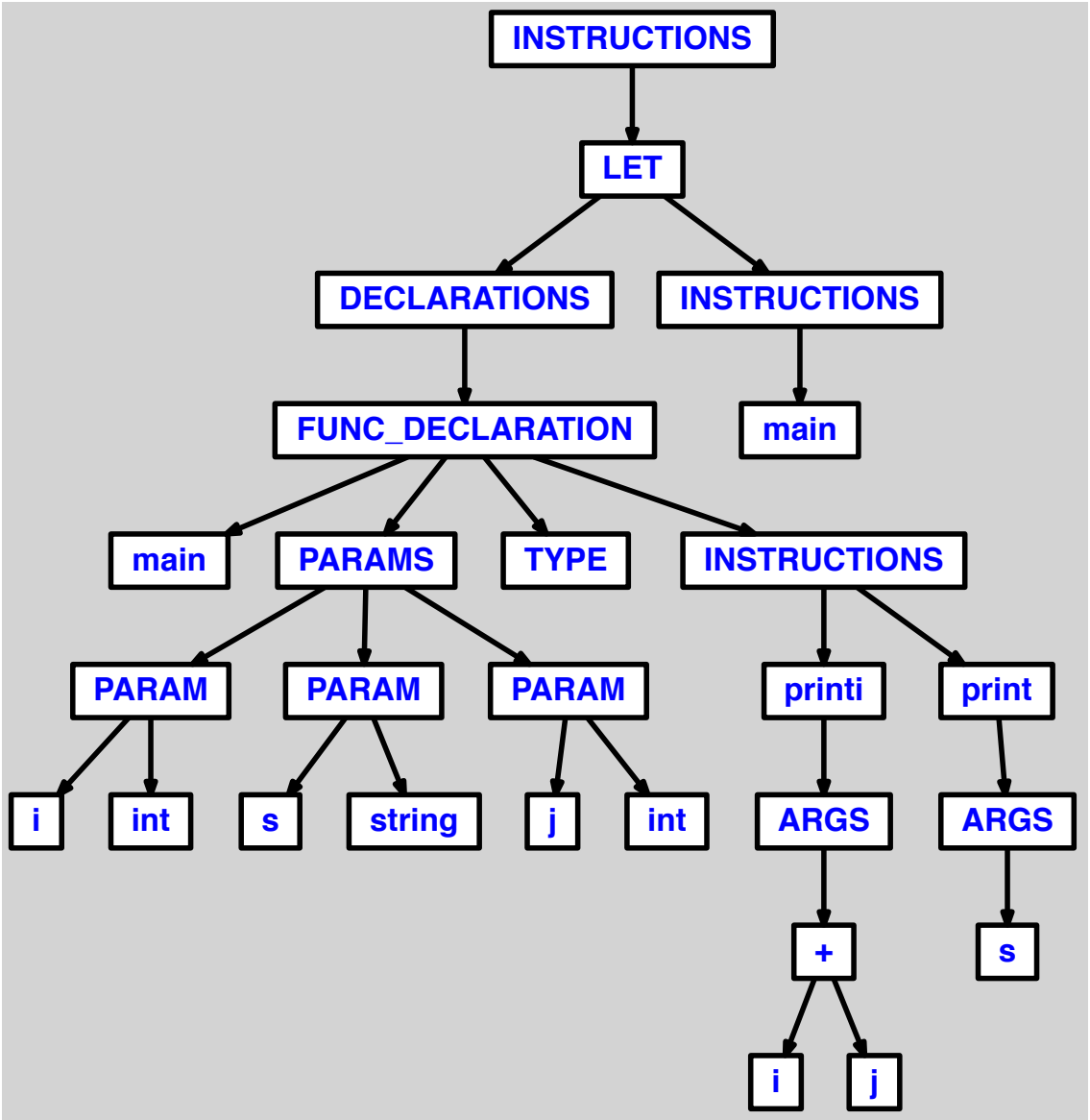
8.2.6 fonction avec 3 parametres entiers

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


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

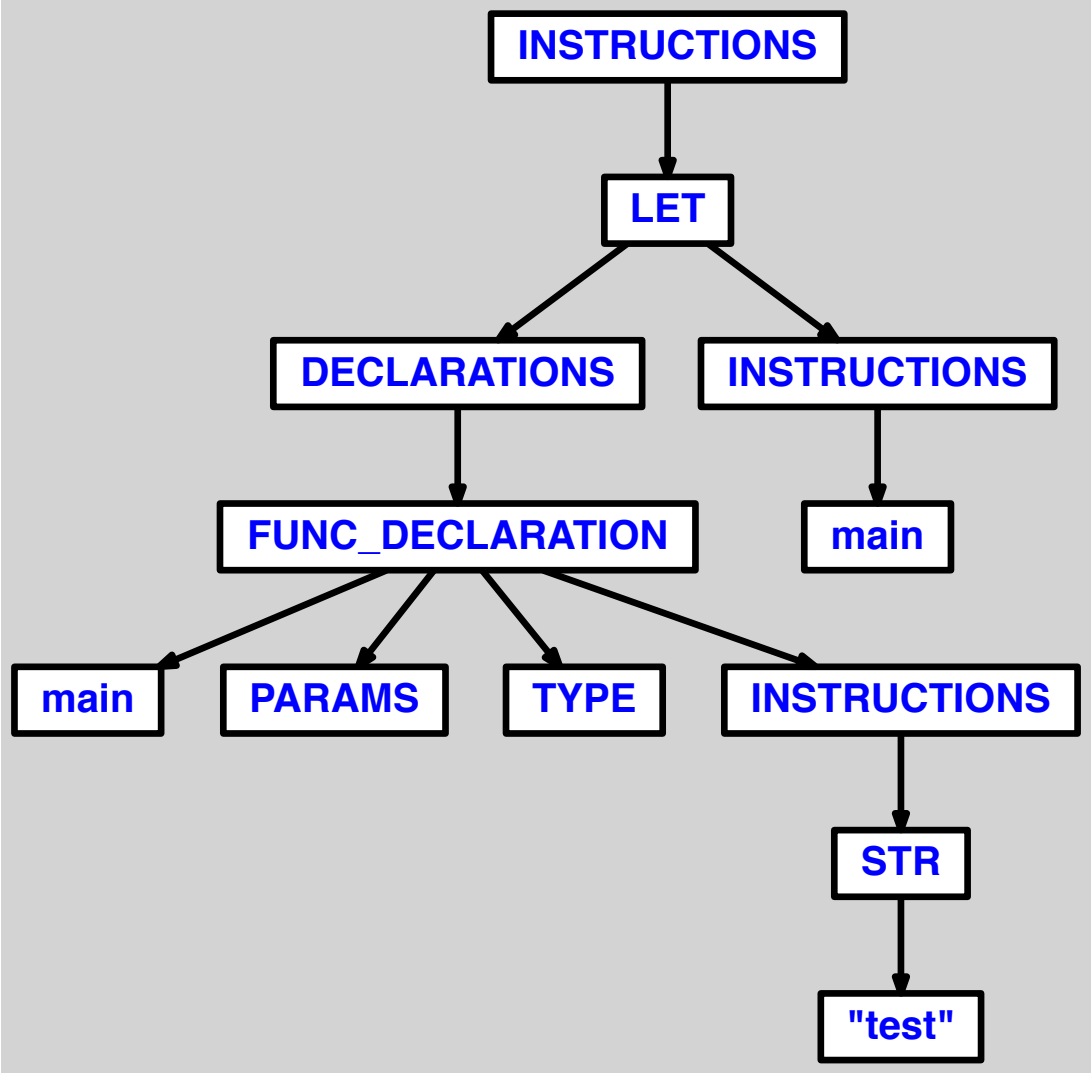
8.2.7 fonction avec 2 parametres entiers et 1 parametre de type <string>

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


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

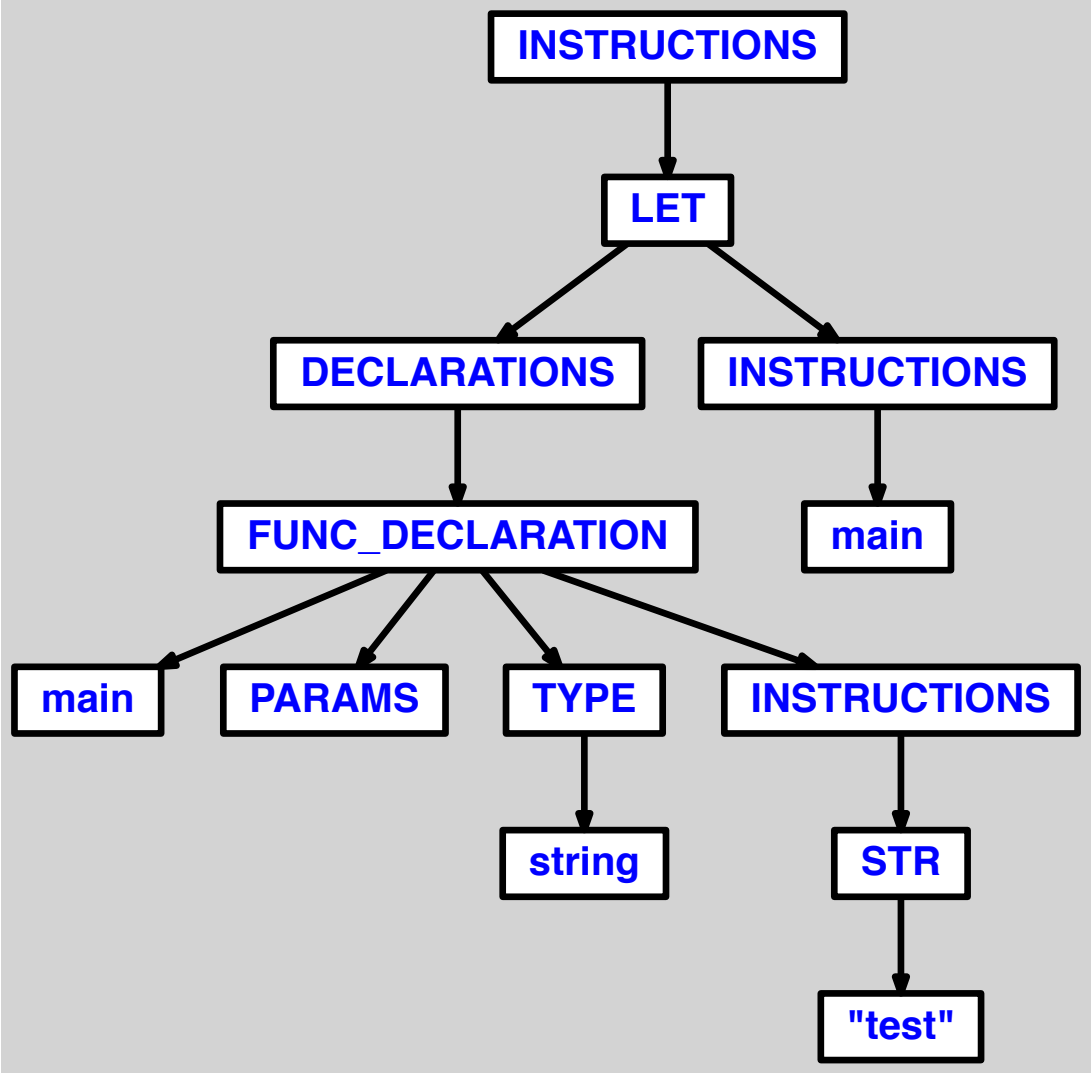
8.2.8 fonction definissant une chaine

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
STR
"test"

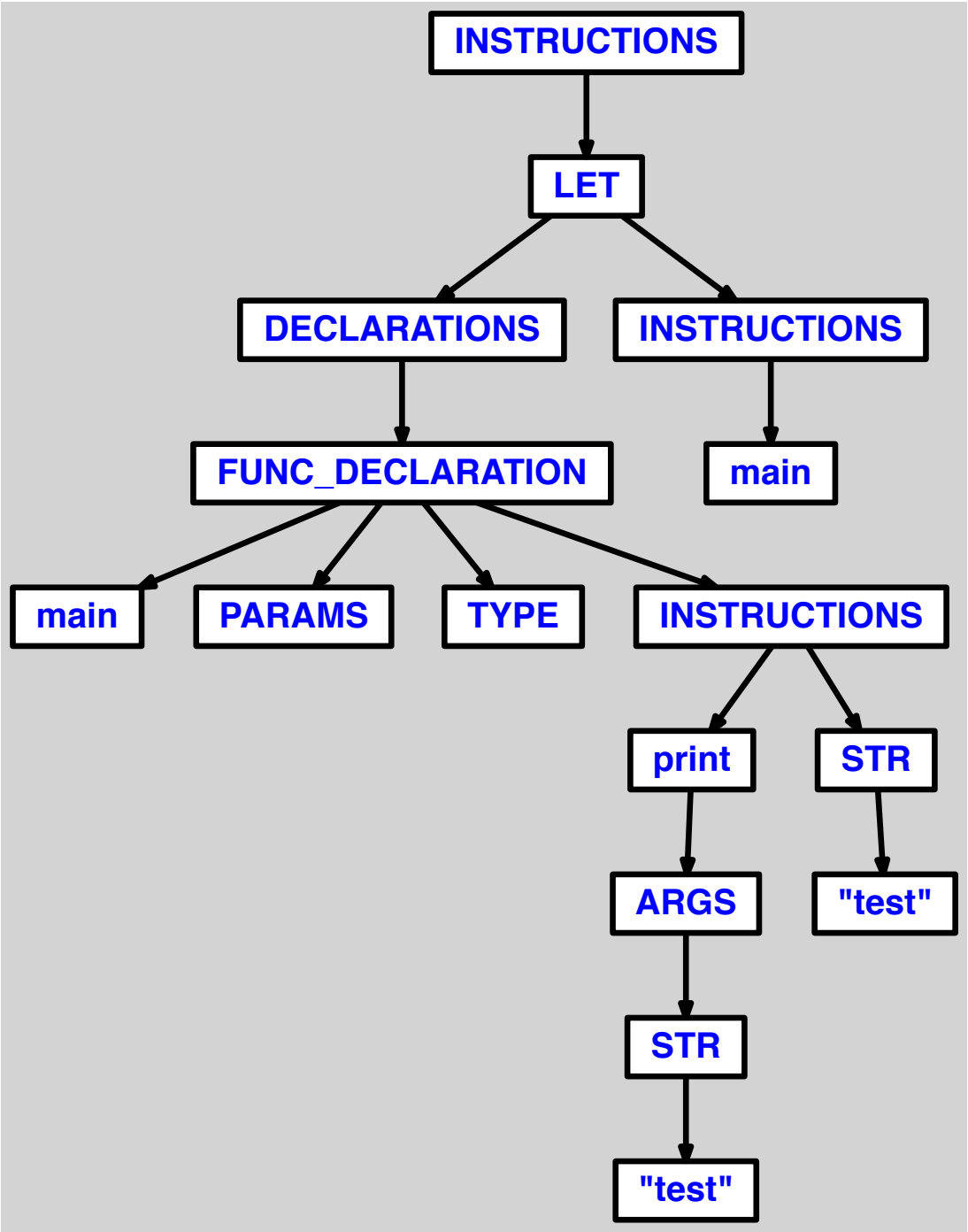
8.2.9 fonction simple, avec declaration du type de retour <string> et instruction de retour

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
string
STR
"test"

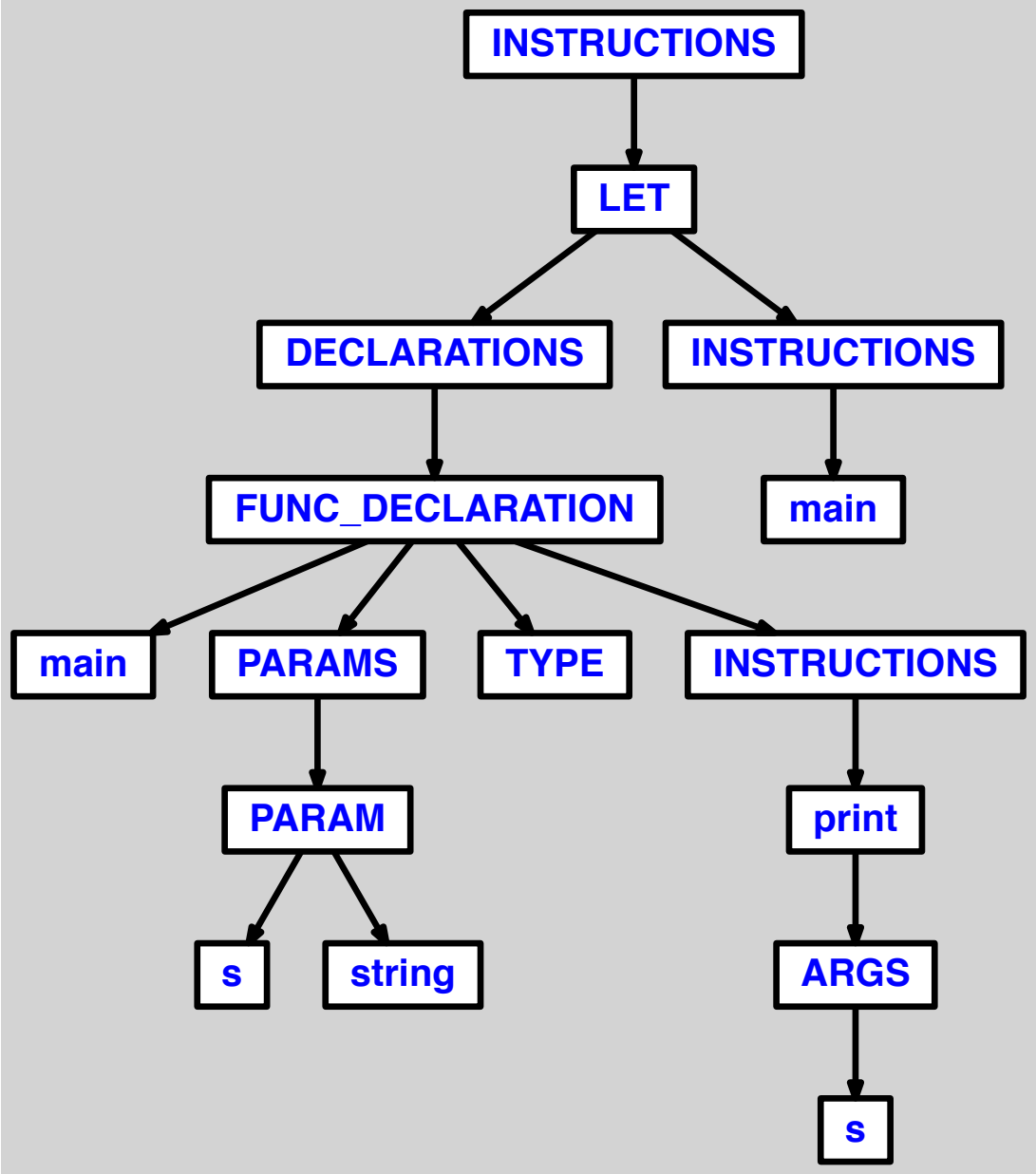
8.2.10 fonction simple, avec instruction simple et instruction retournant une chaine

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


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

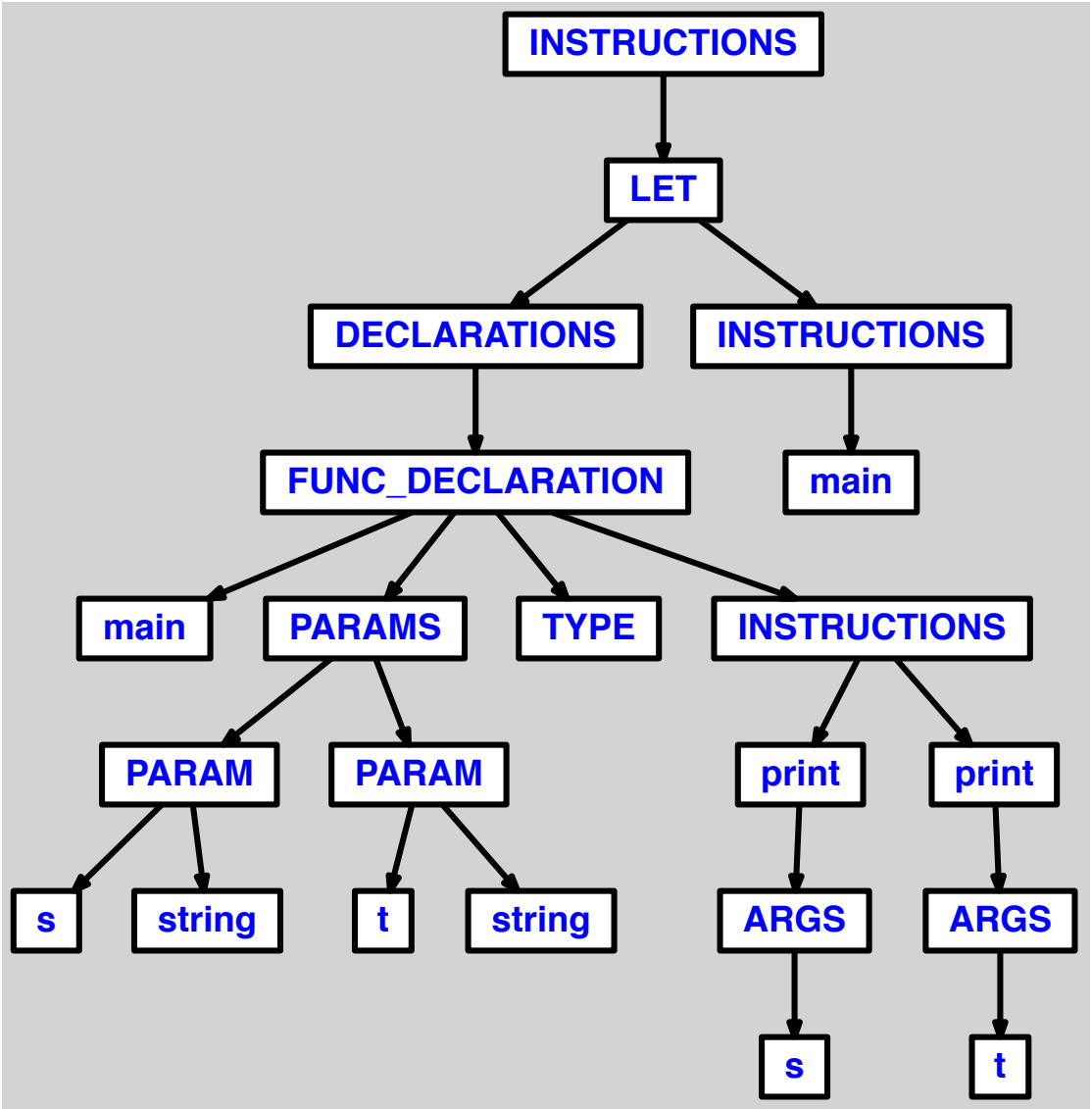
8.2.11 fonction avec 1 parametre de type <string>

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


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

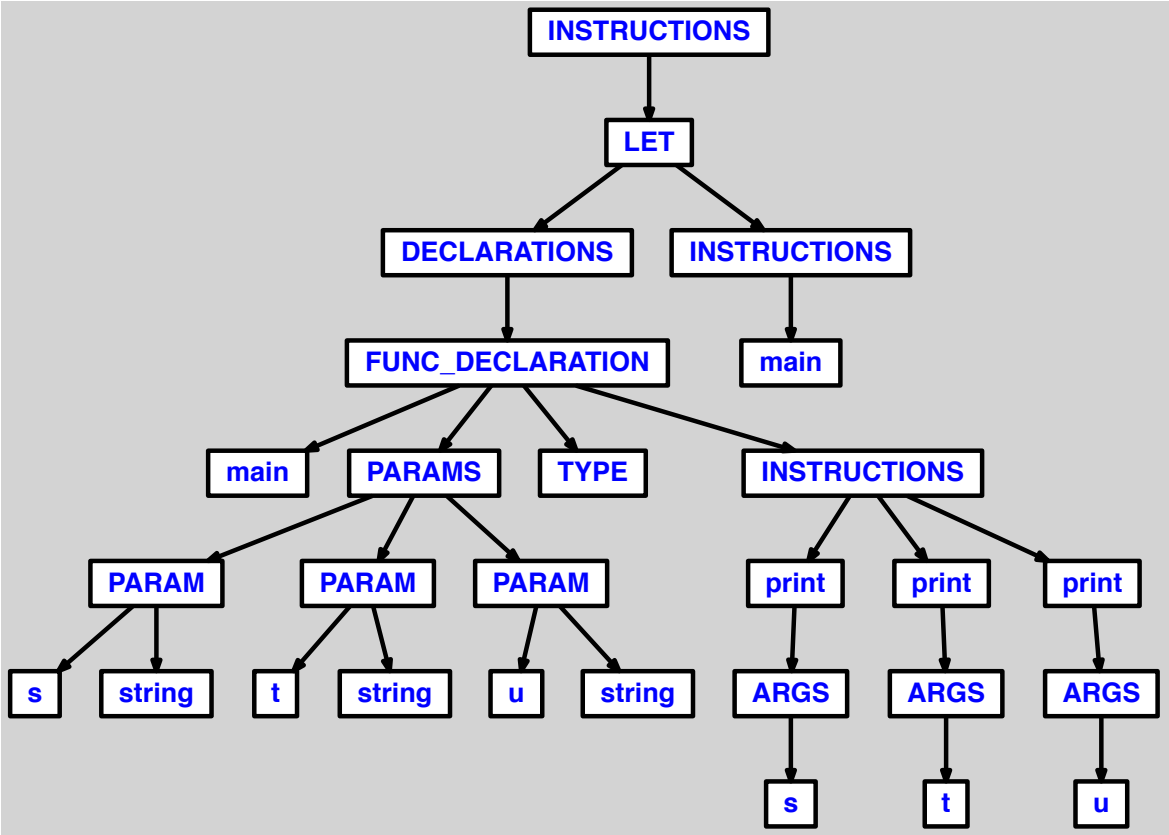
8.2.12 fonction avec 2 parametres de type <string>

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


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

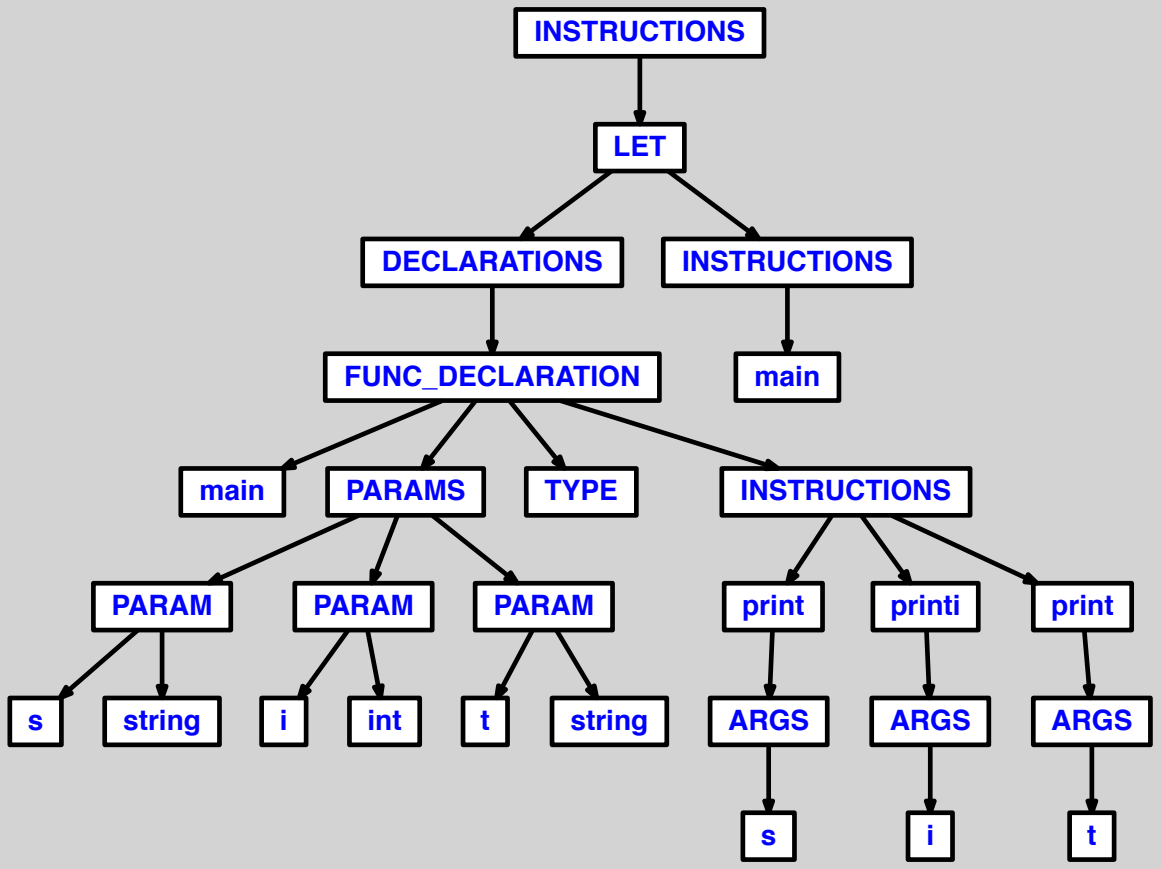
8.2.13 fonction avec 3 parametres de type <string>

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


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

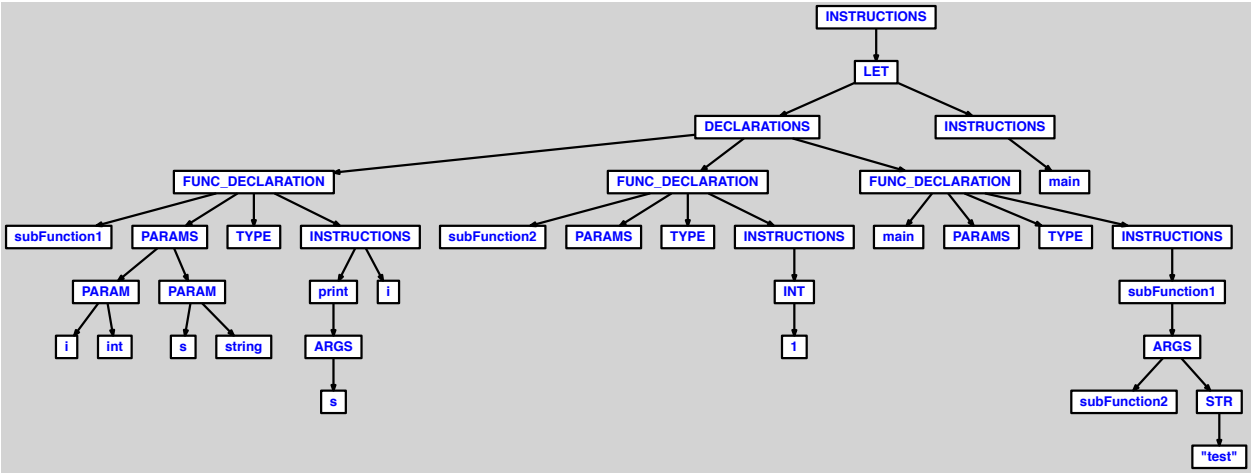
8.2.14 fonction avec 2 parametres de type <string> et 1 parametre entier

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


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

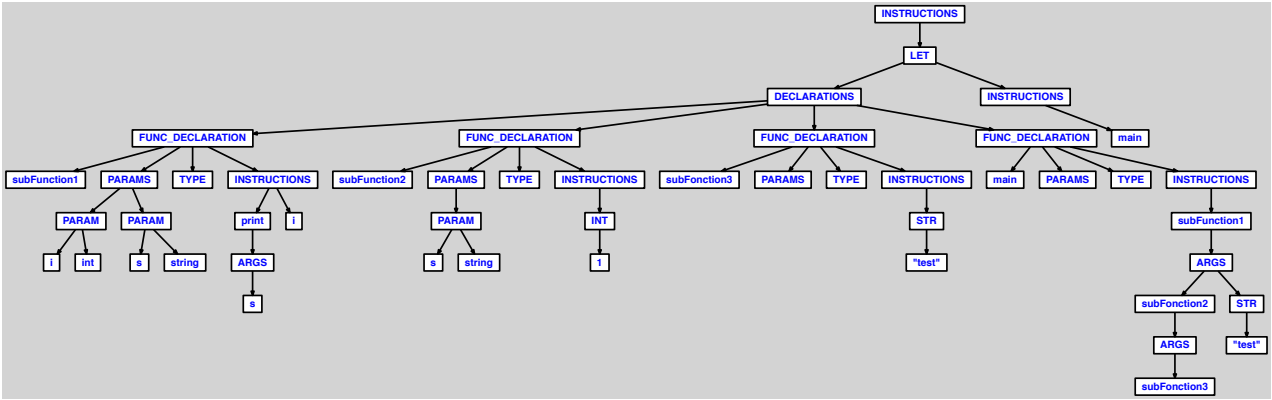
8.2.15 double-imbrication de fonction

```
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
main
subFunction1
PARAMS
TYPE
INSTRUCTIONS
subFunction2
PARAMS
TYPE
INSTRUCTIONS
main
PARAMS
TYPE
INSTRUCTIONS
PARAM
PARAM
print
i
INT
subFunction1
i
int
s
string
ARGS
1
ARGS
s
subFunction2
STR
"test"

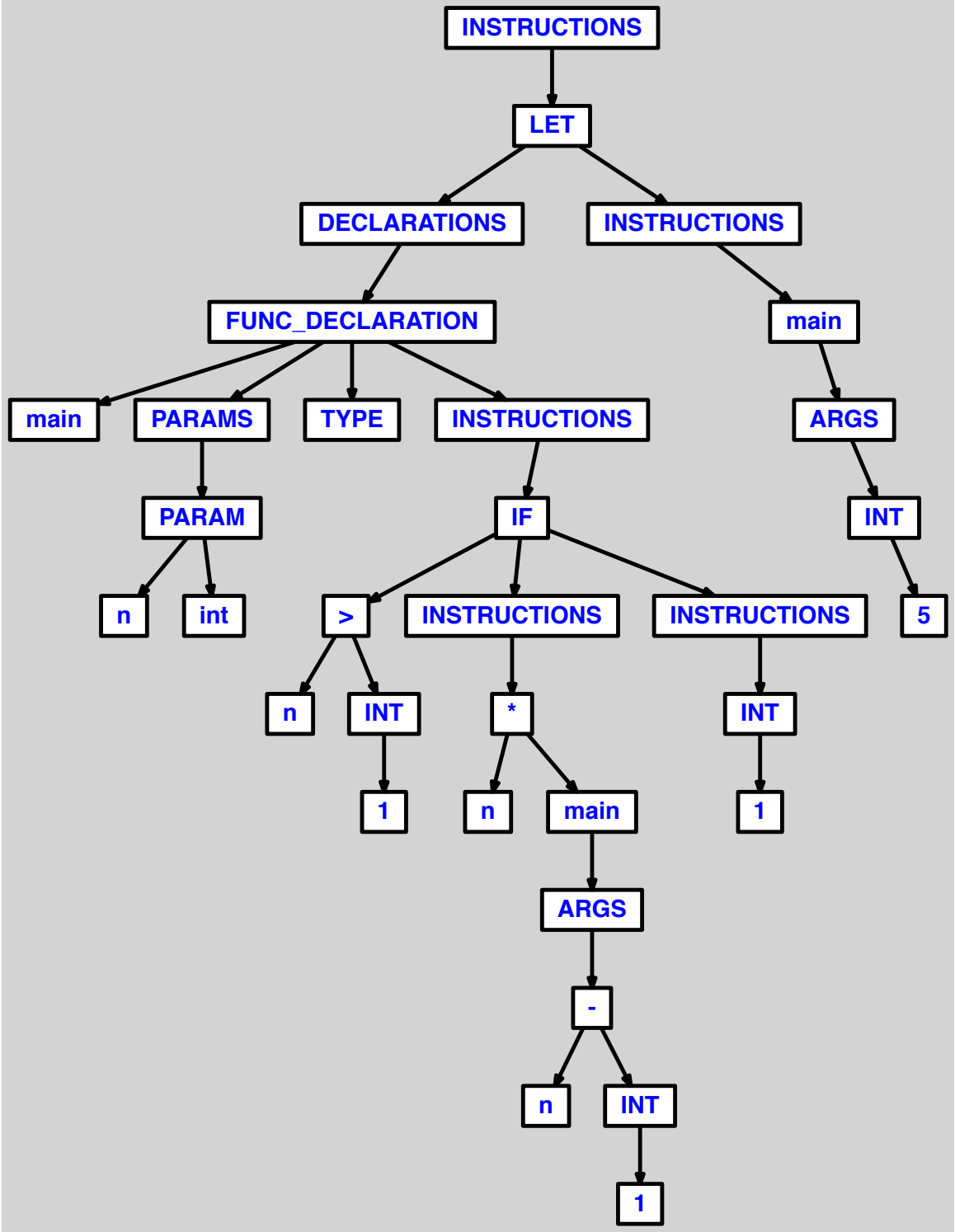
8.2.16 triple-imbrication de fonction

```
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
main
subFunction1
PARAMS
TYPE
INSTRUCTIONS
subFunction2
PARAMS
TYPE
INSTRUCTIONS
subFonction3
PARAMS
TYPE
INSTRUCTIONS
main
PARAMS
TYPE
INSTRUCTIONS
PARAM
PARAM
print
i
PARAM
INT
STR
subFunction1
i
int
s
string
ARGS
s
string
1
"test"
ARGS
s
subFonction2
STR
ARGS
"test"
subFonction3

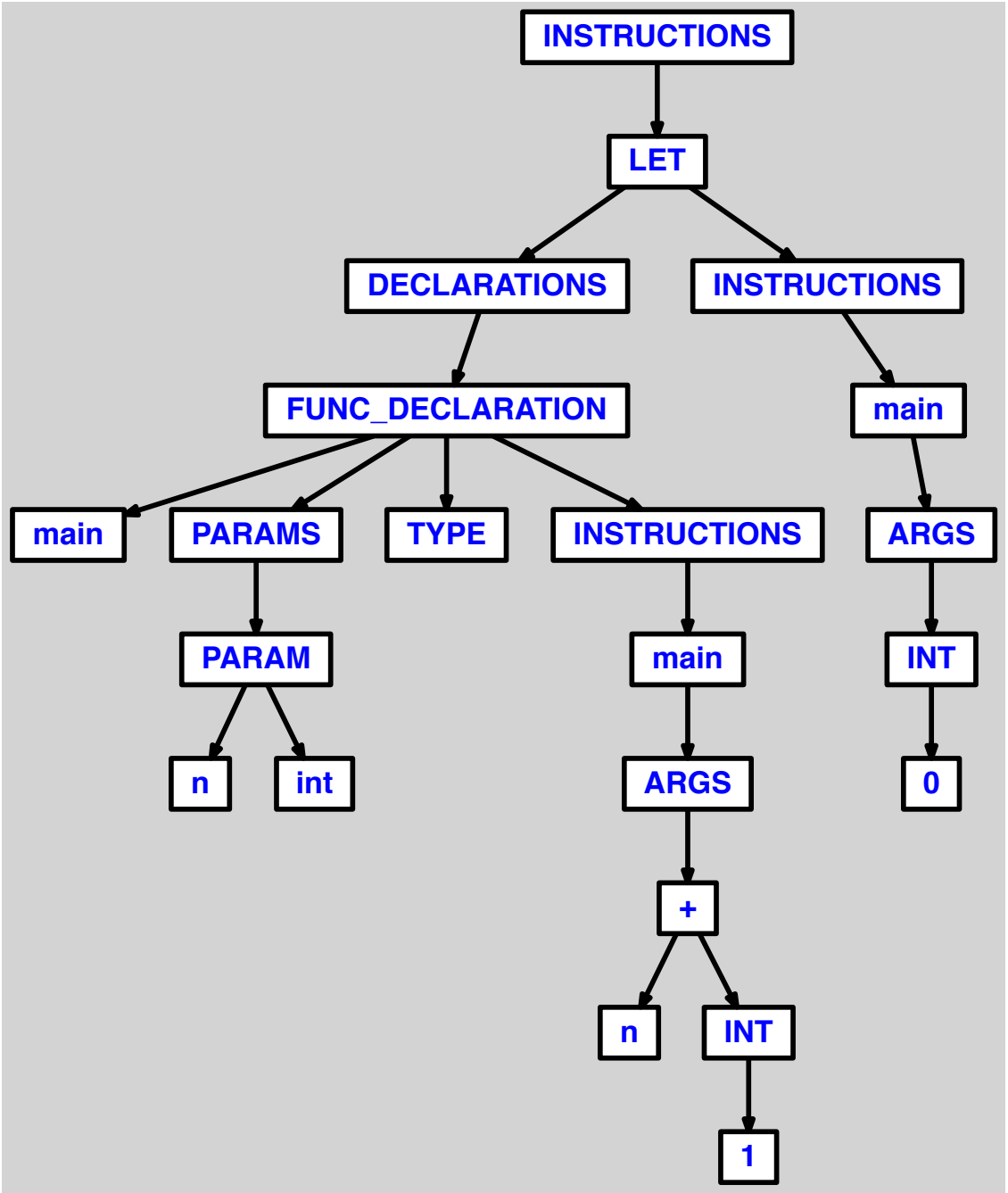
8.2.17 recursion finie

```
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
main
main
PARAMS
TYPE
INSTRUCTIONS
ARGS
PARAM
IF
INT
n
int
>
INSTRUCTIONS
INSTRUCTIONS
5
n
INT
*
INT
1
n
main
1
ARGS
-
n
INT
1

8.2.18 recursion infinie

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

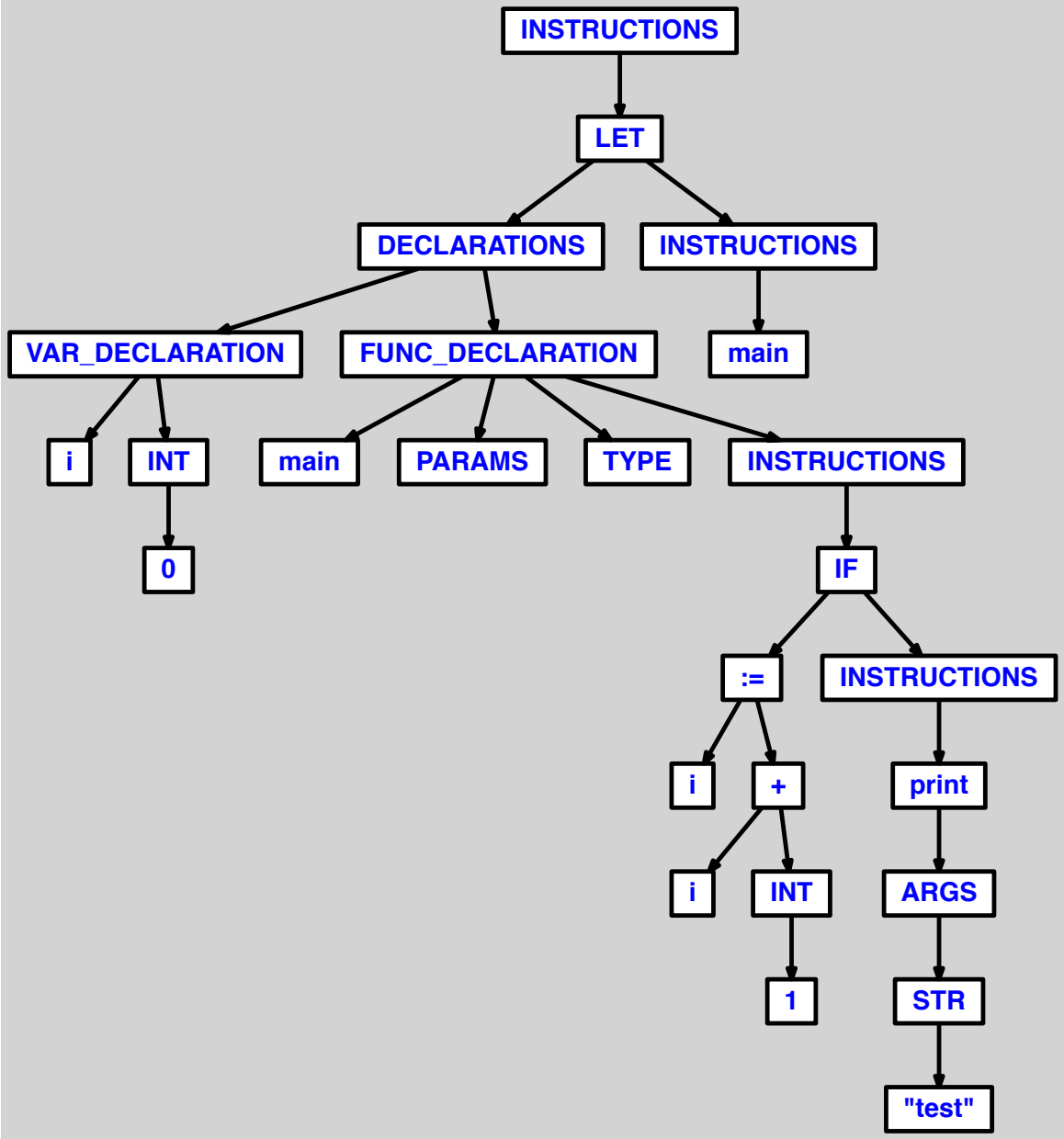

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
ARGS
PARAM
main
INT
n
int
ARGS
0
+
n
INT
1

9 if

9.1 KO

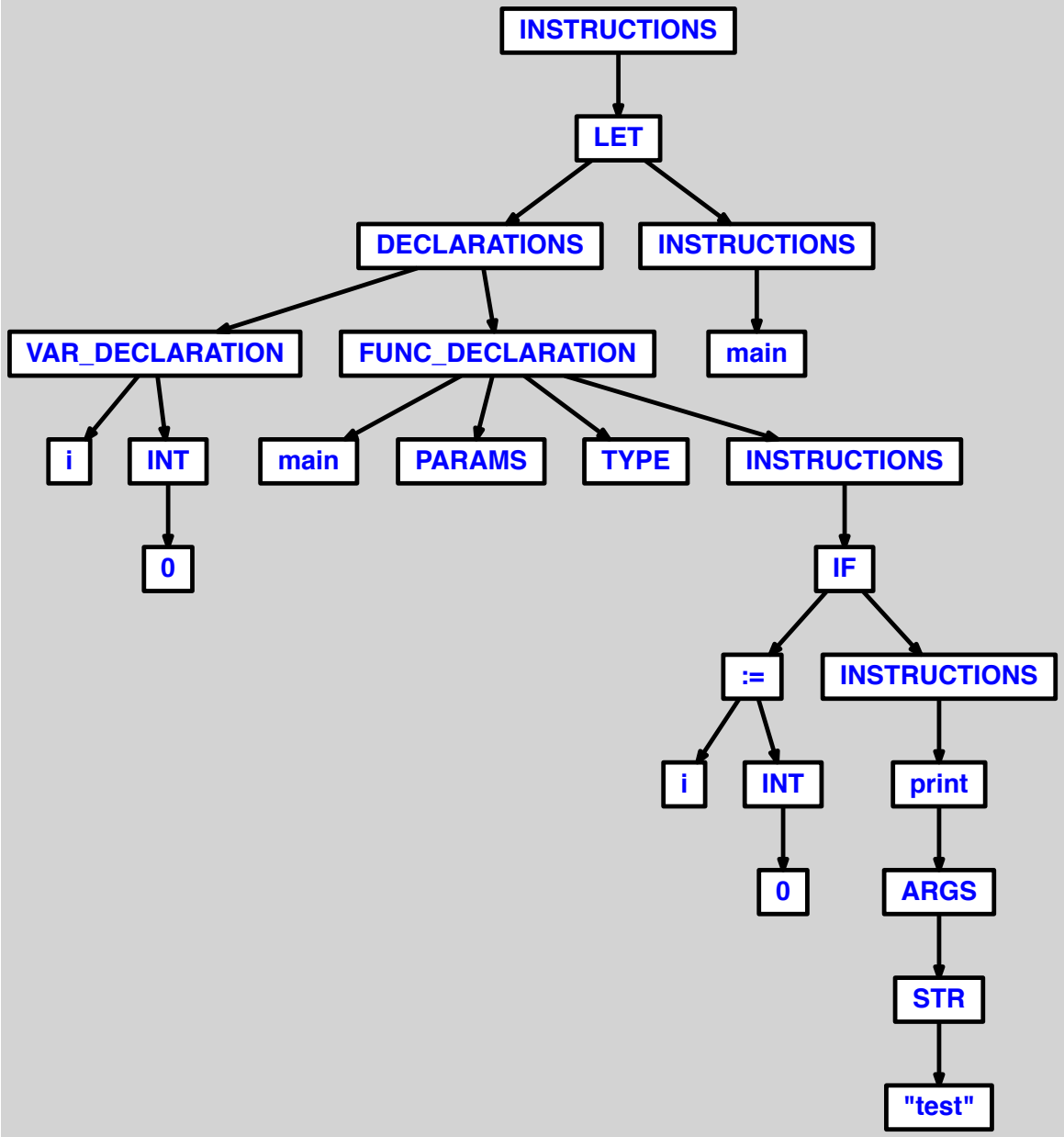
9.1.1 incrementation dans la condition <if then>

```
let
  var i := 0

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


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

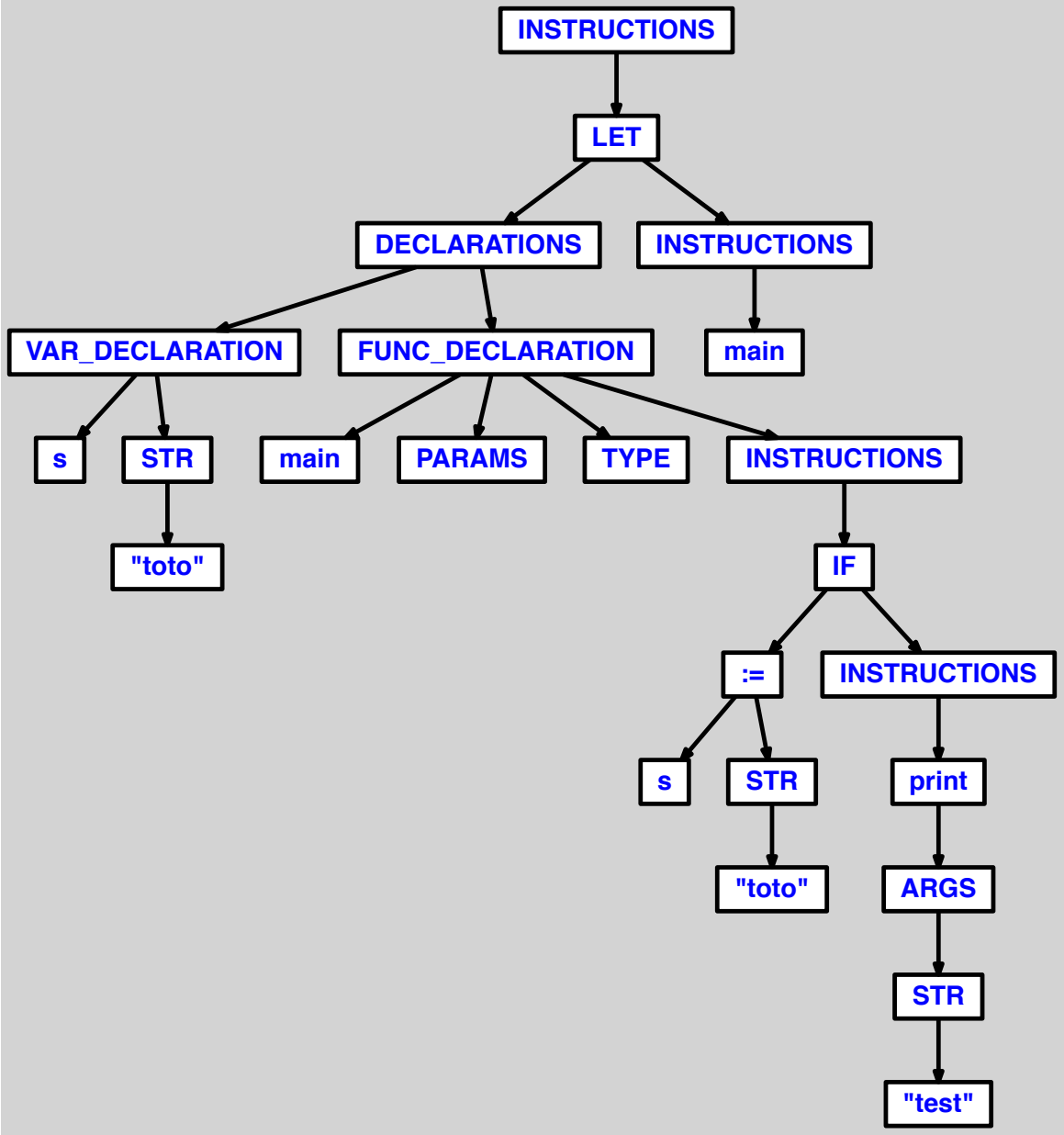
9.1.2 affectation d'entier dans la condition <if then>

```
let
  var i := 0

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


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

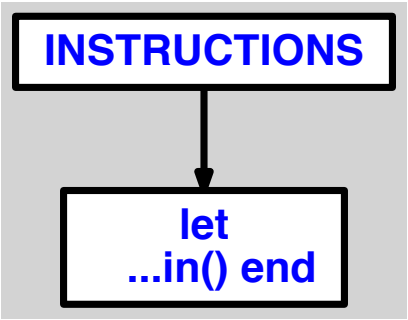
9.1.3 affectation de chaine dans la condition <if then>

```
let
  var s := "toto"

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


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

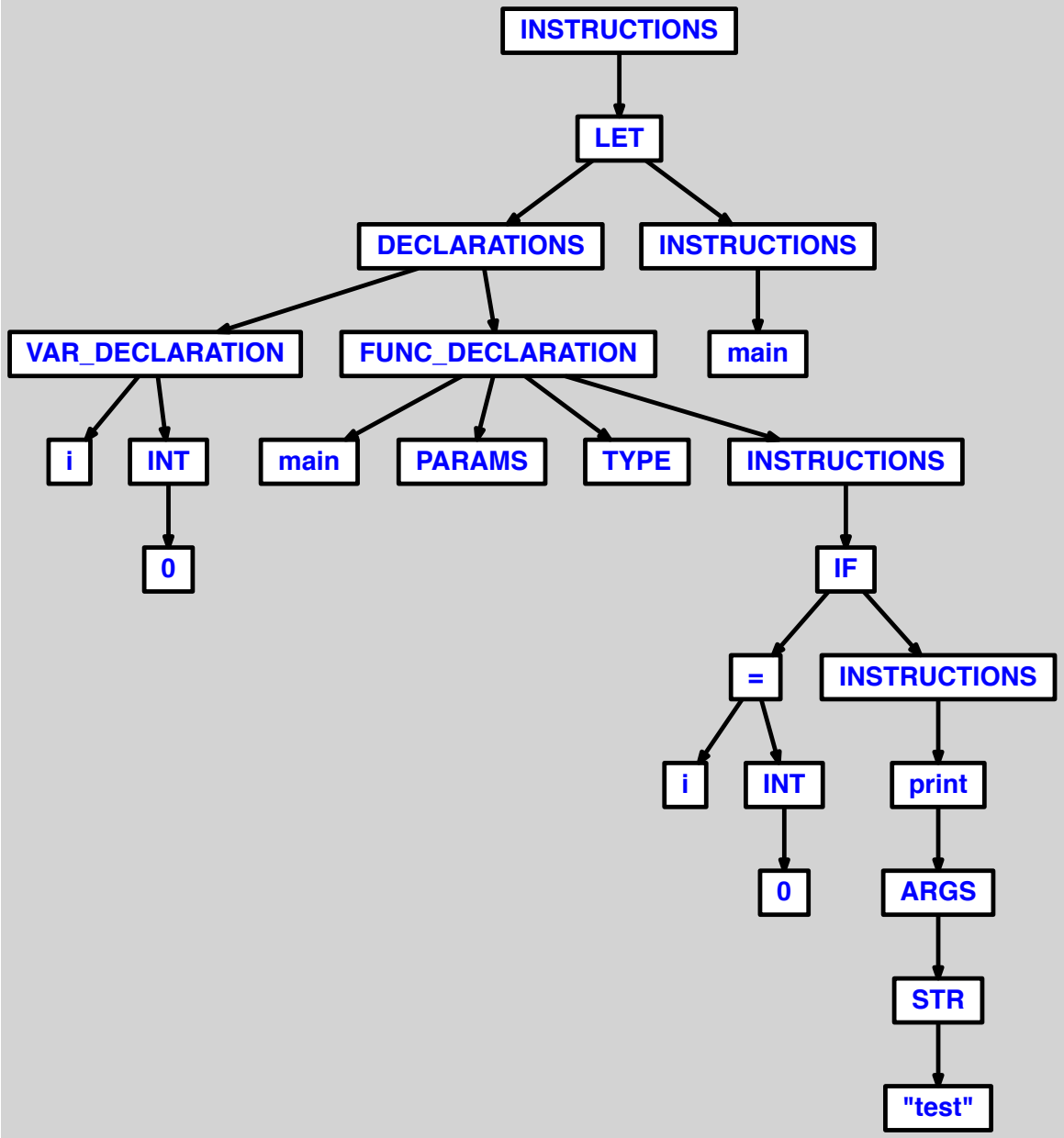
9.1.4 <if then> avec oubli du <if>

```
let
  var i := 0

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


INSTRUCTIONS
let
...in() end

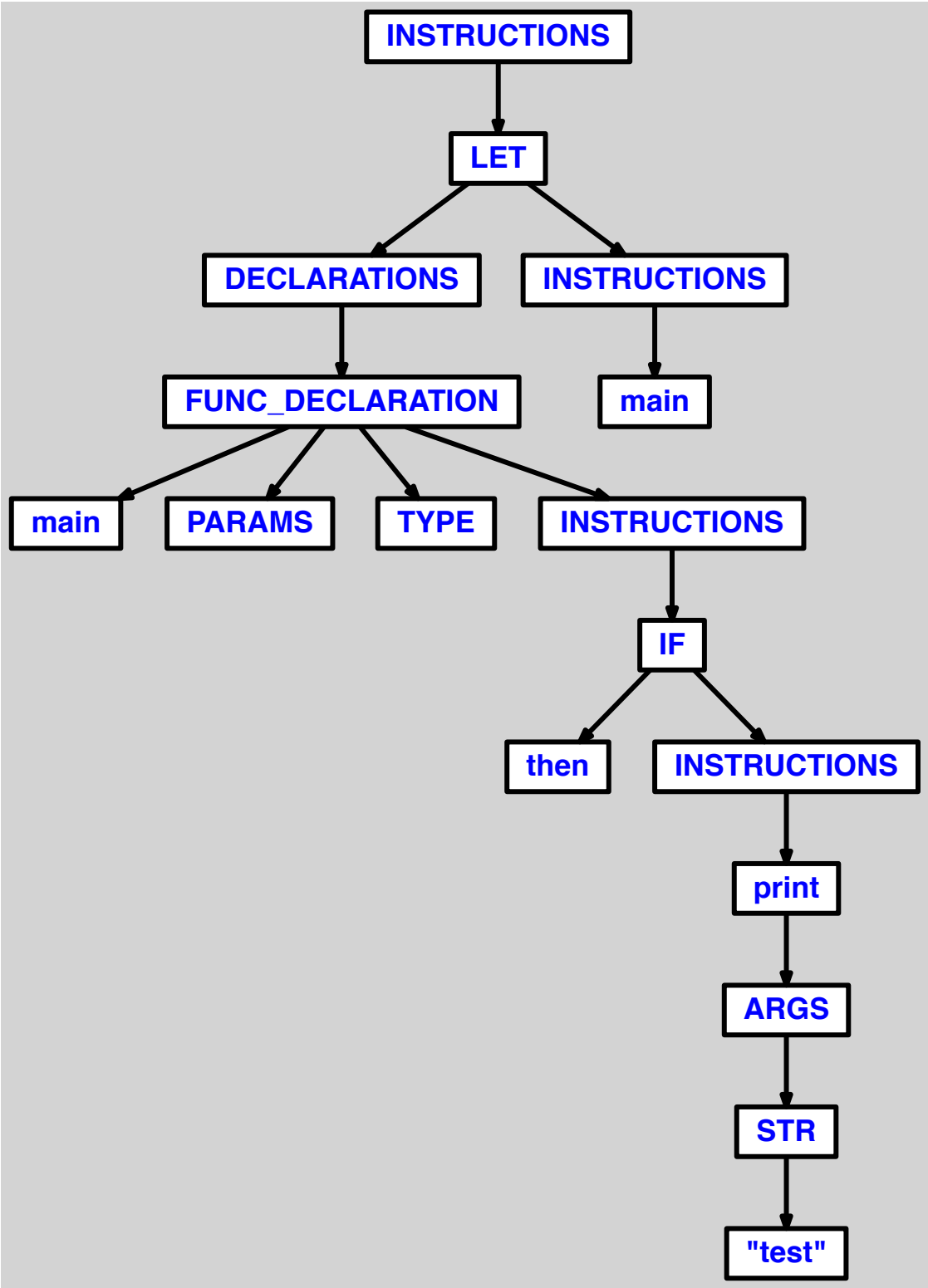
9.1.5 <if then> avec oubli du <then>

```
let
  var i := 0

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


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

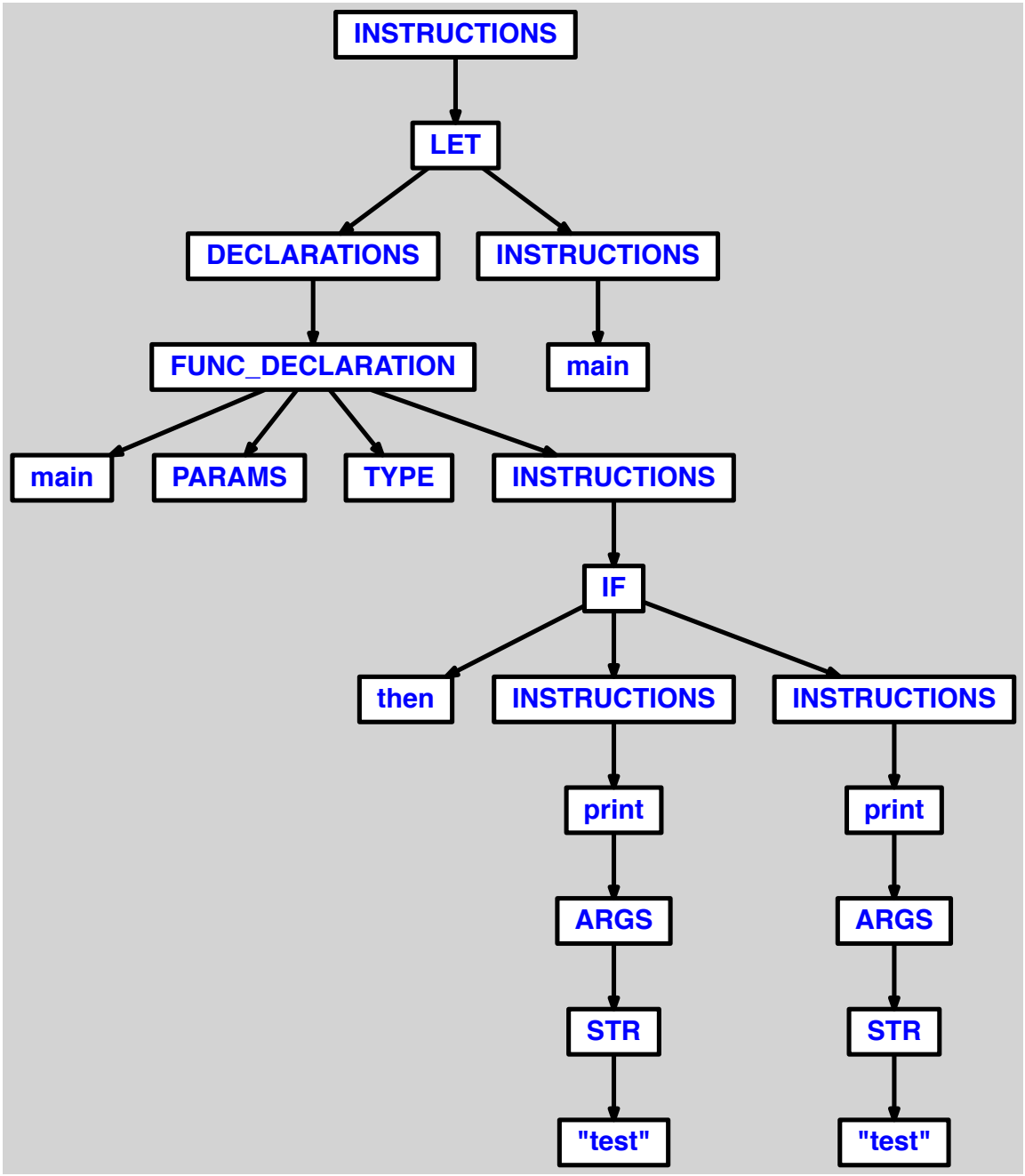
9.1.6 <if then> avec oubli de la condition

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


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

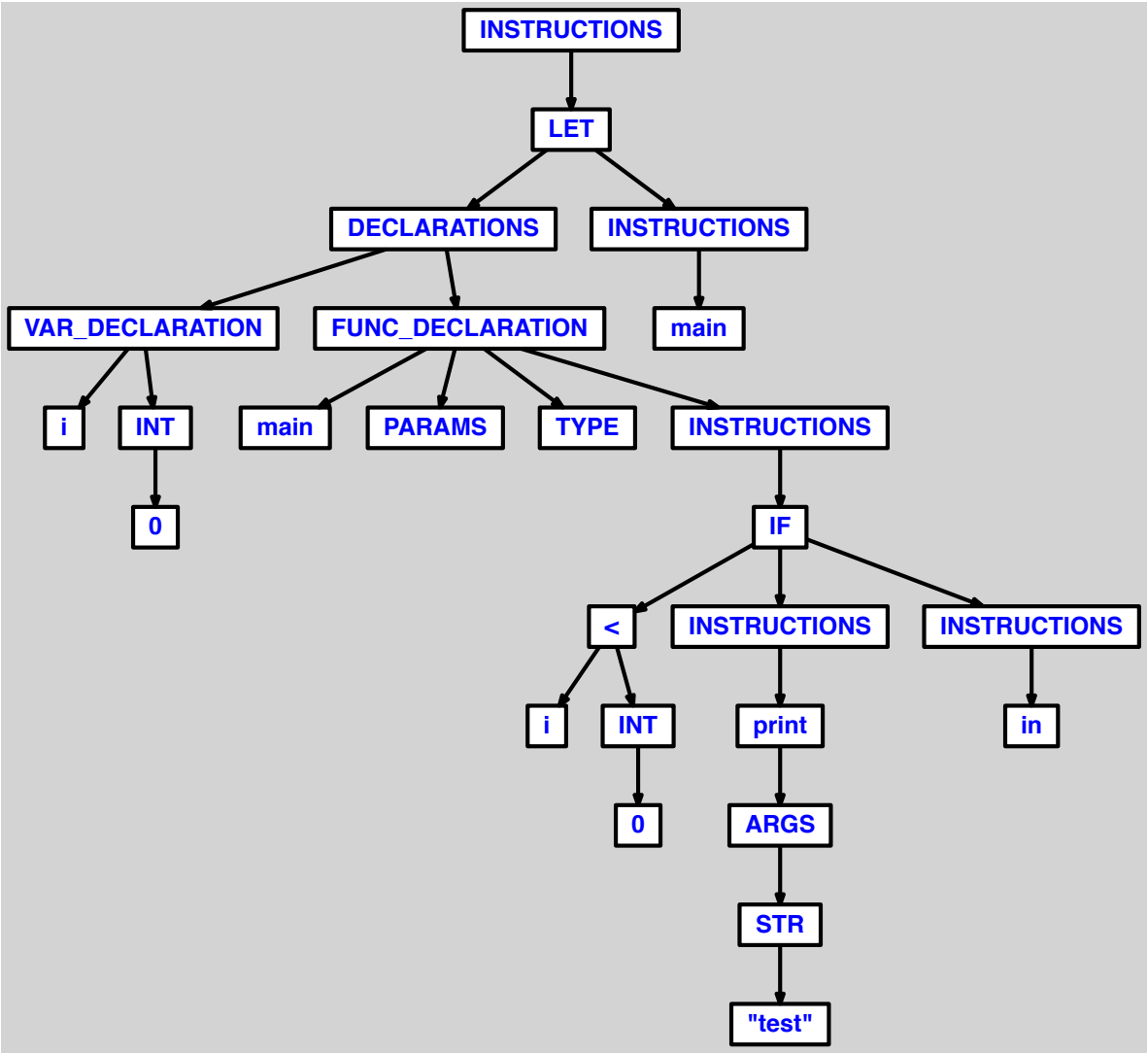
9.1.7 <if then else> avec oubli de la condition

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


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

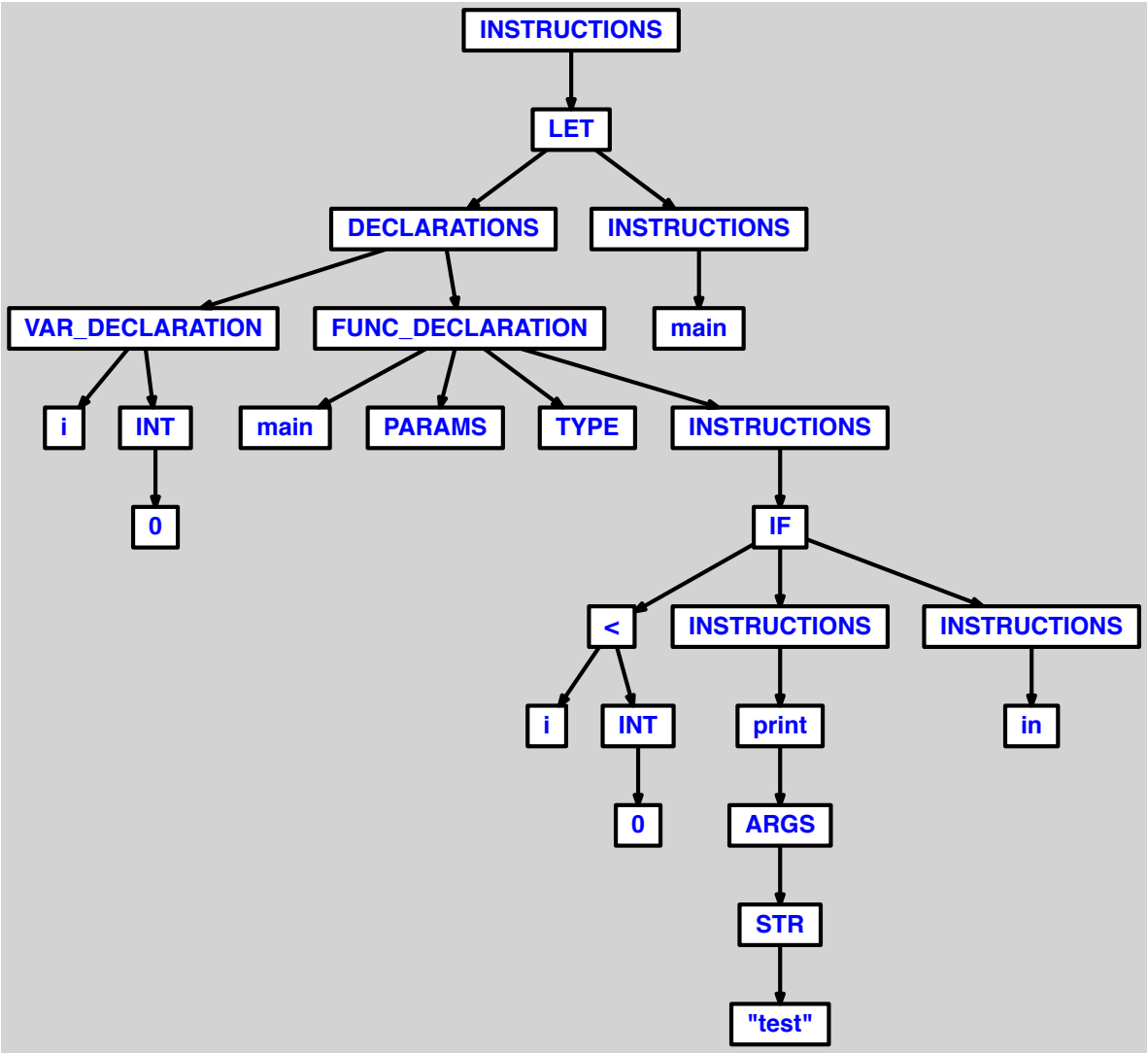
9.1.8 <if then else> sans instruction dans <else>

```
let
  var i := 0

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


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

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

```
let
  var i := 0

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

in main() end
```


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

9.1.10 <if then> avec condition entiere

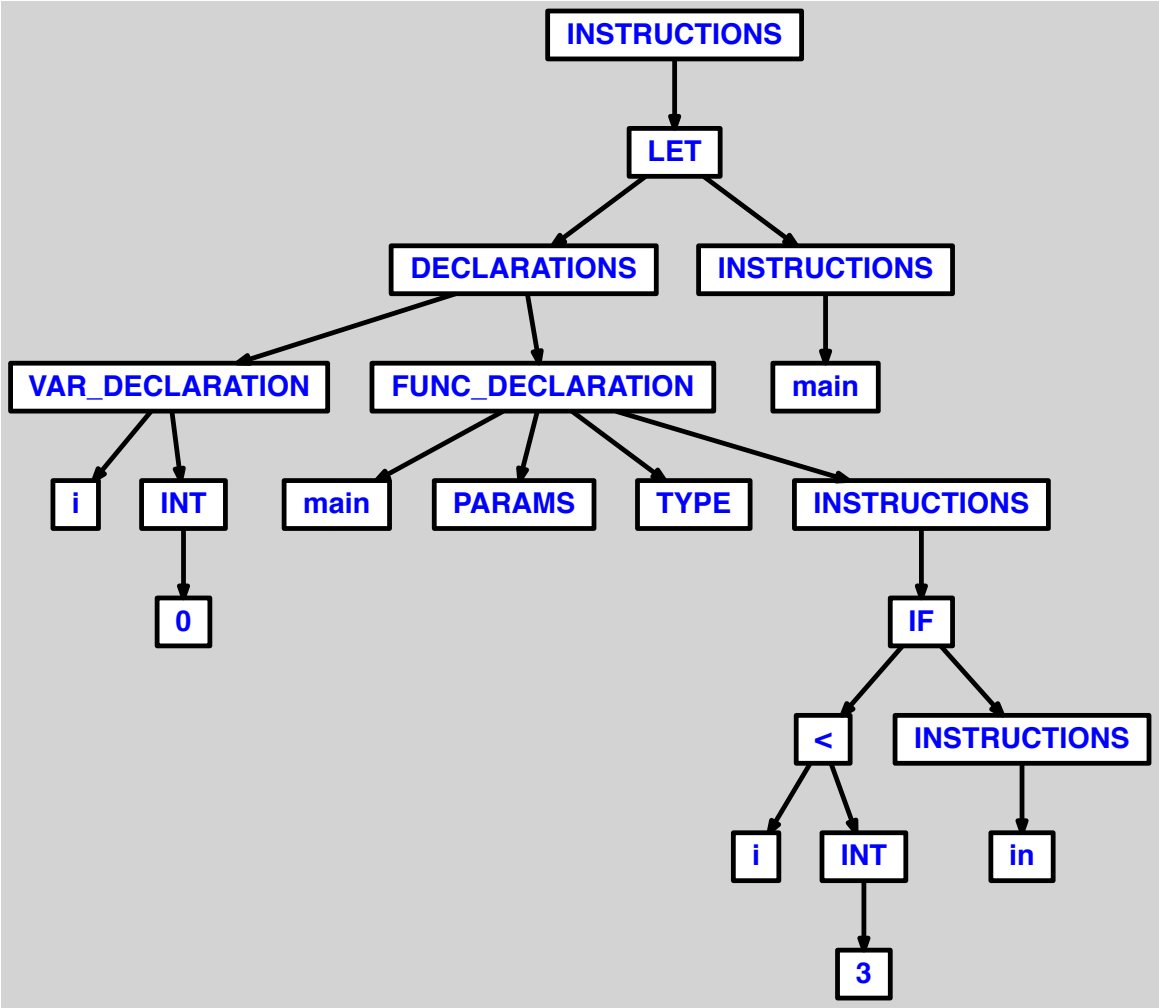
```
let
  function main()() =
    if 1 then
      print("test")
in main() end
```

Pas d'AST, problème de syntaxe.

9.2 OK

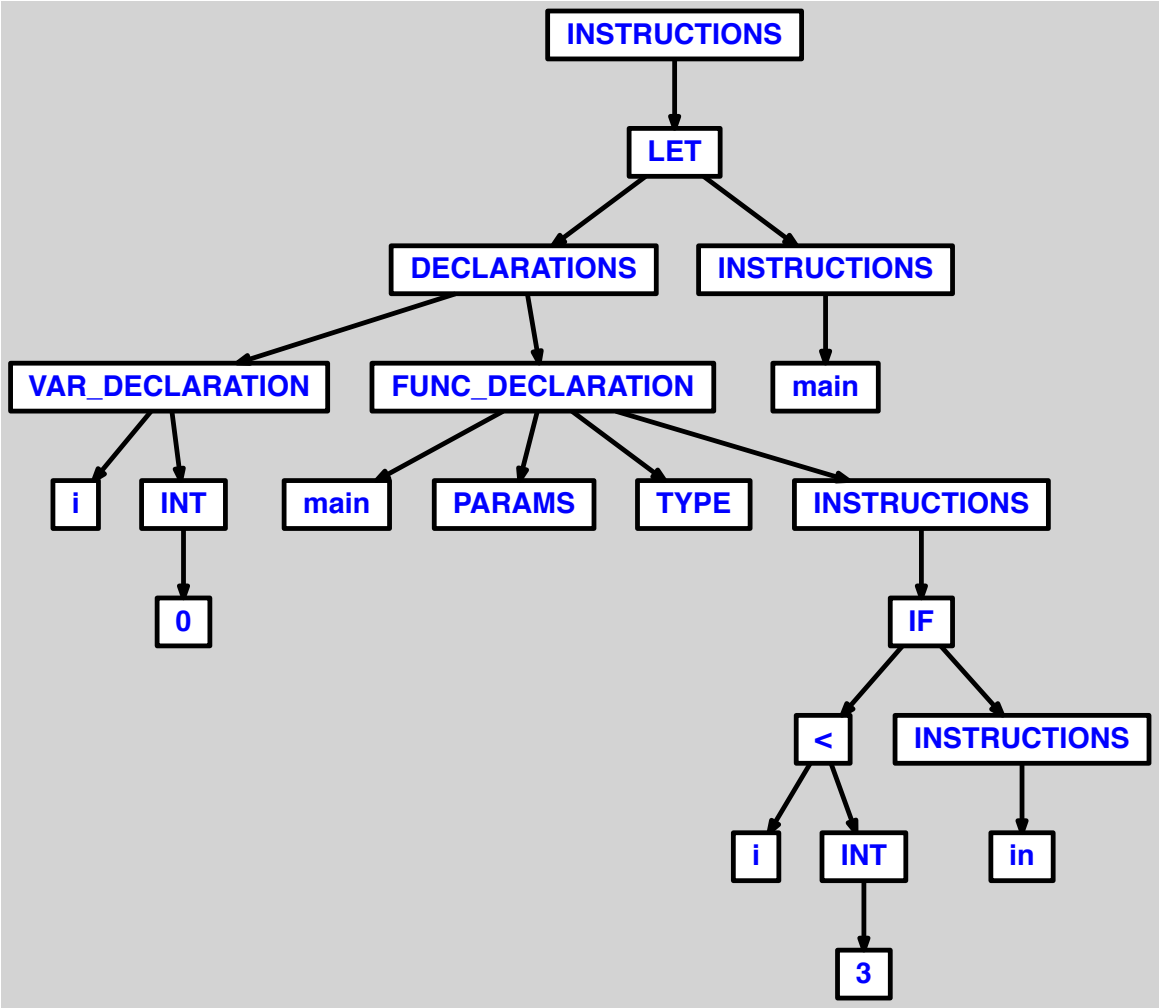
9.2.1 <if then> sans instruction

```
let
  var i := 0

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


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

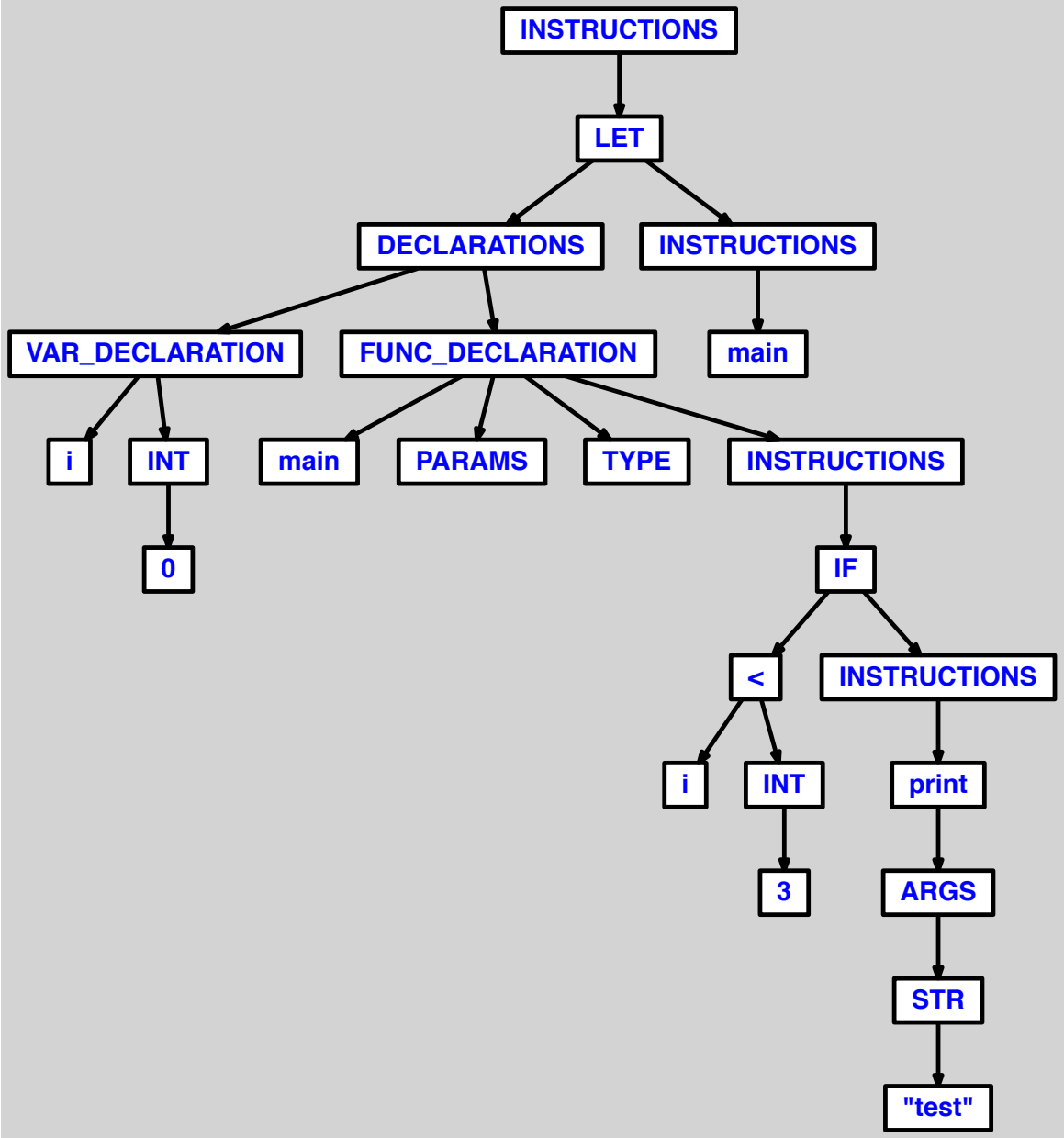
9.2.2 <if then> avec ligne vide

```
let
  var i := 0

  function main() =
    if i < 3 then

in main() end
```


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

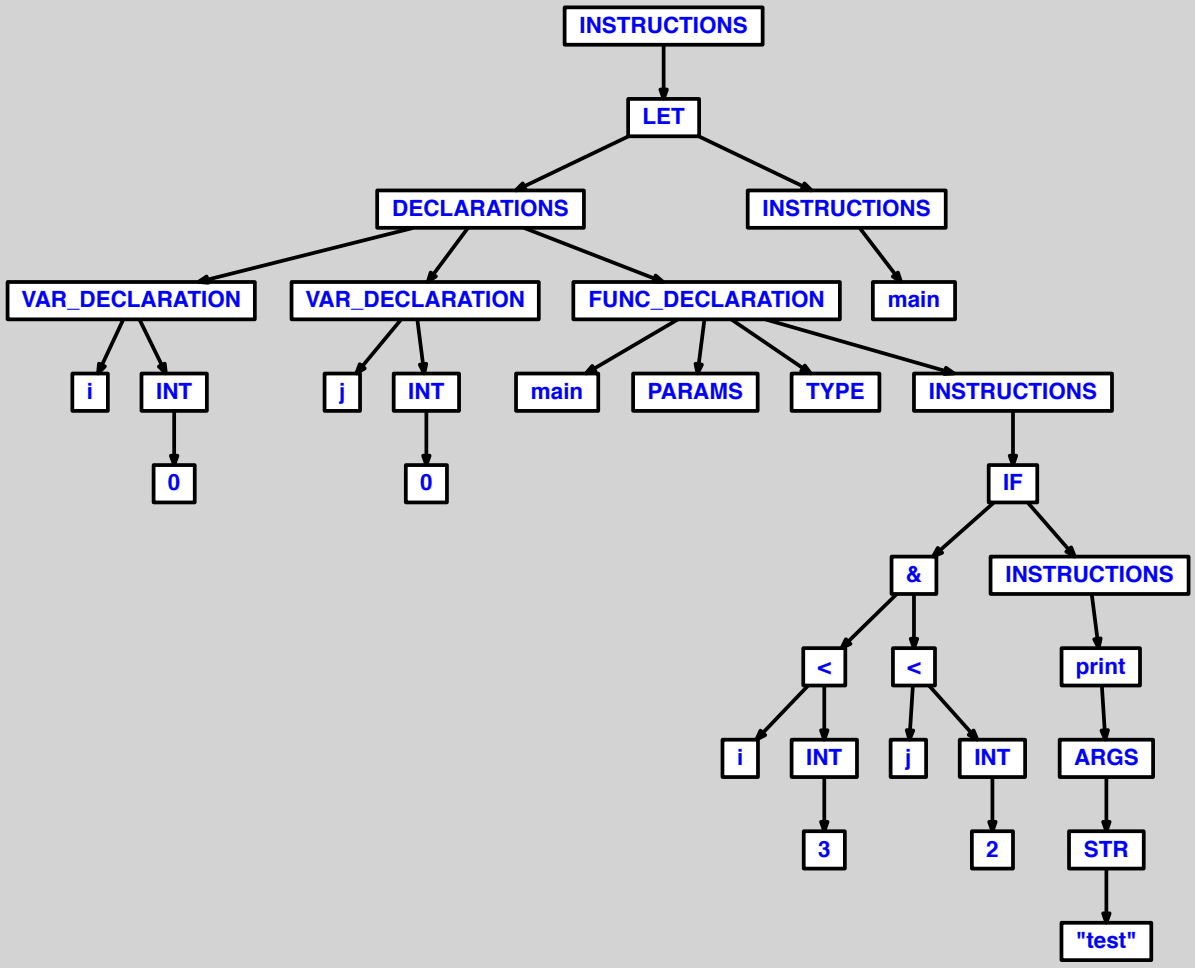
9.2.3 <if then> avec simple condition

```
let
  var i := 0

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


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

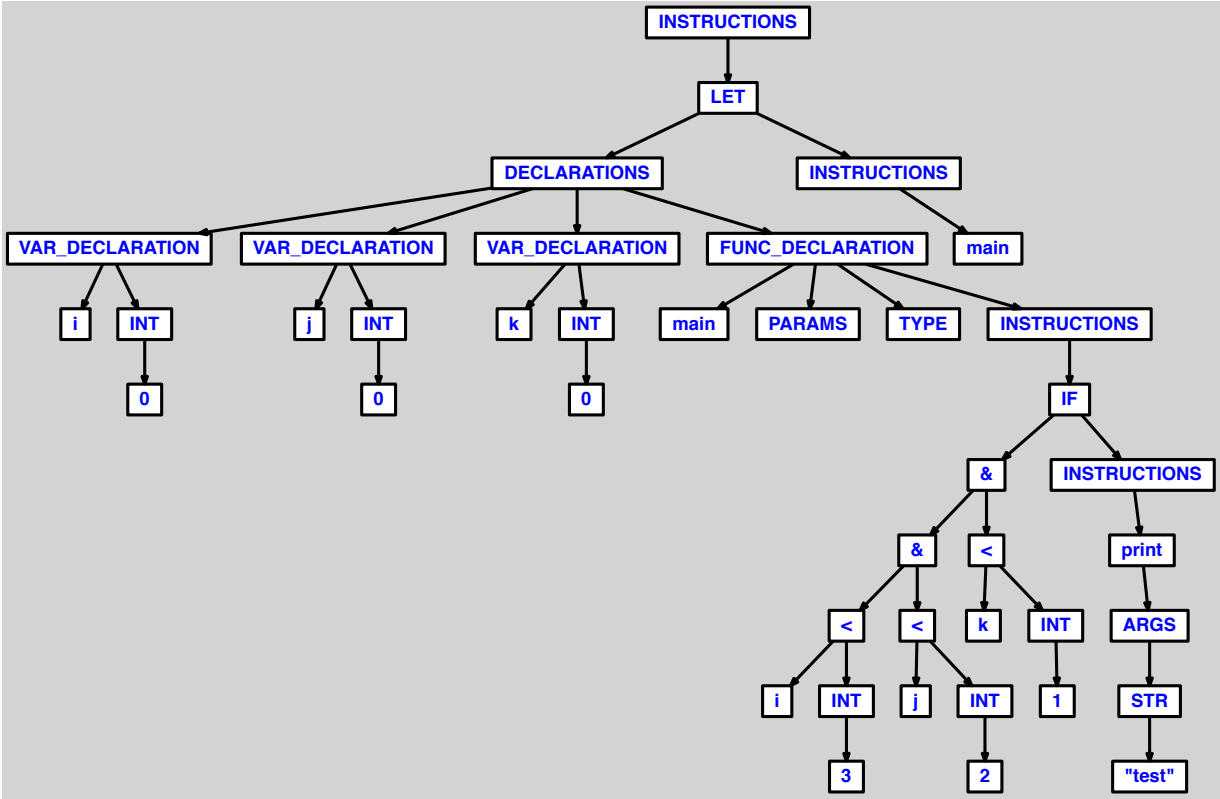
9.2.4 <if then> avec double-condition

```
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
main
i
INT
j
INT
main
PARAMS
TYPE
INSTRUCTIONS
0
0
IF
&
INSTRUCTIONS
<
<
print
i
INT
j
INT
ARGS
3
2
STR
"test"

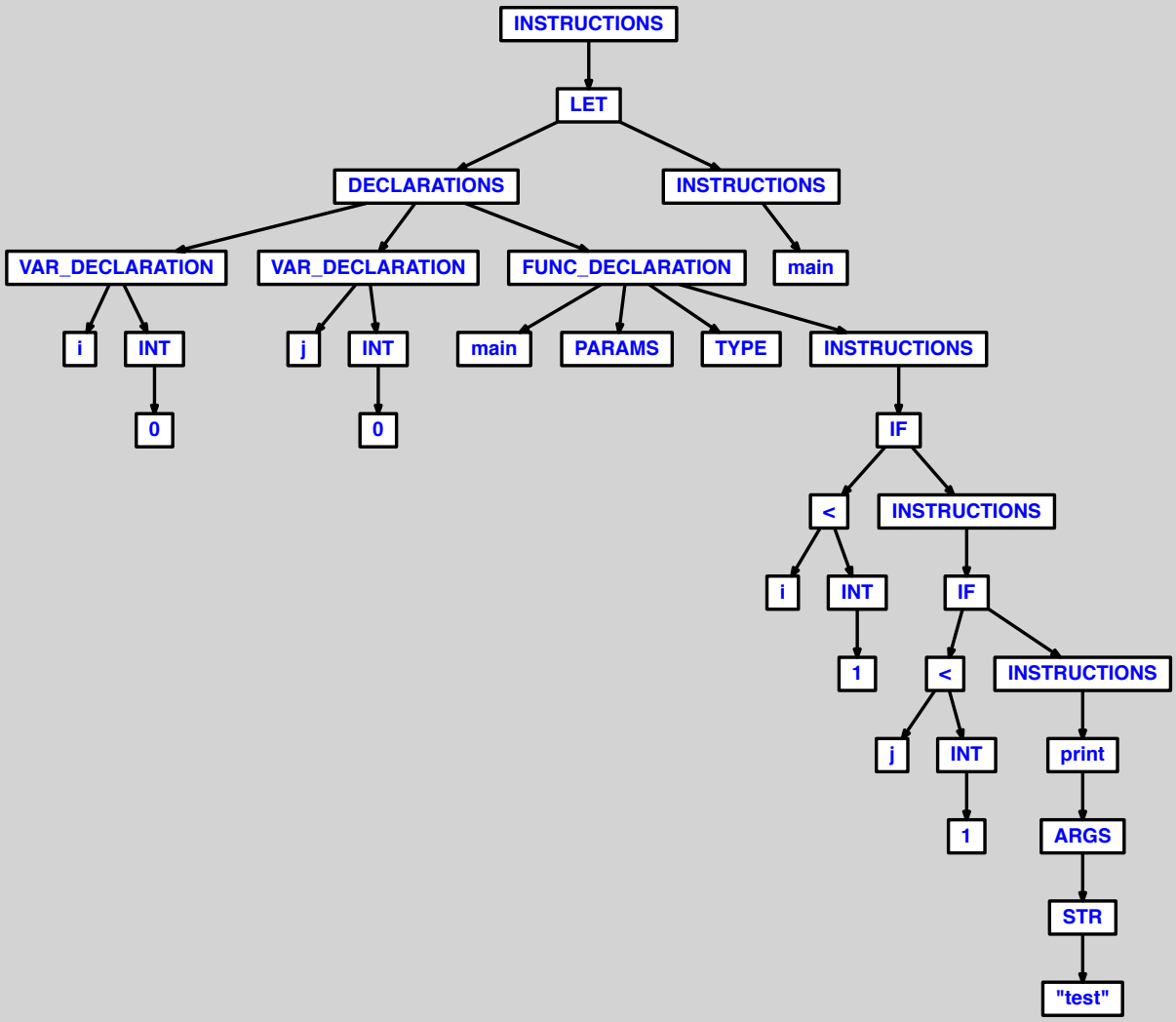
9.2.5 <if then> avec triple-condition

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
0
0
0
IF
&
INSTRUCTIONS
&
<
print
<
<
k
INT
ARGS
i
INT
j
INT
1
STR
3
2
"test"

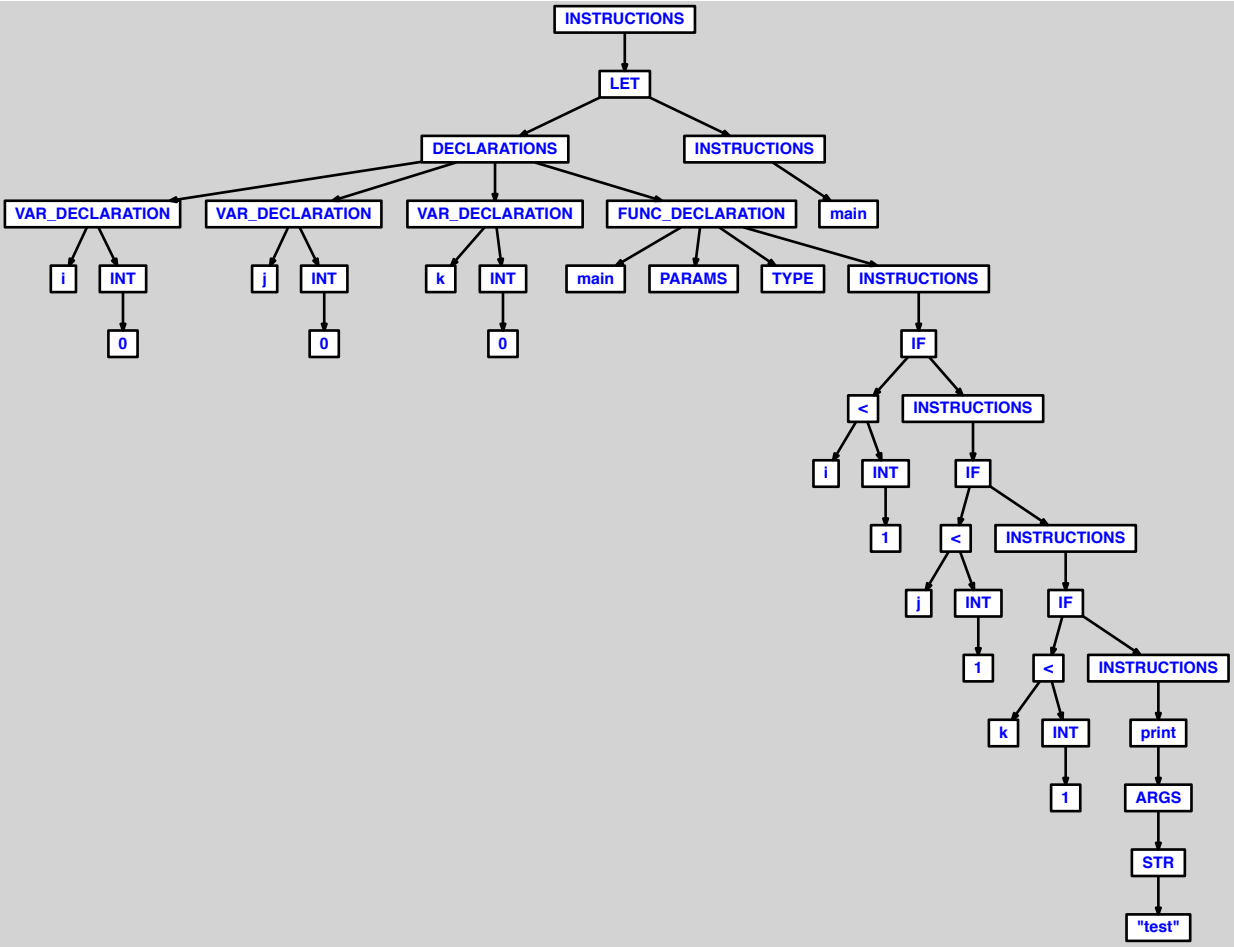
9.2.6 double-imbrication de <if then>

```
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
main
i
INT
j
INT
main
PARAMS
TYPE
INSTRUCTIONS
0
0
IF
<
INSTRUCTIONS
i
INT
IF
1
<
INSTRUCTIONS
j
INT
print
1
ARGS
STR
"test"

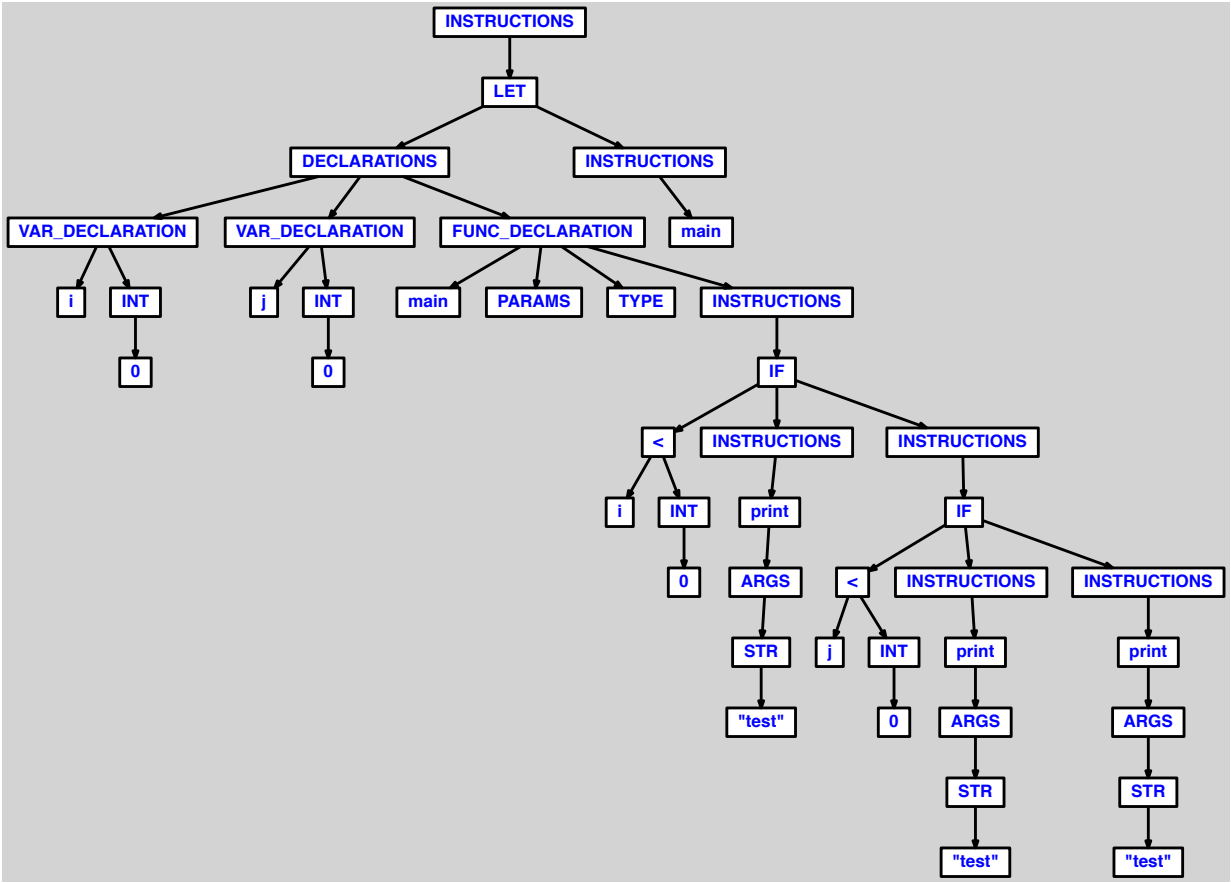
9.2.7 triple-imbrication de <if then>

```
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
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
0
0
0
IF
<
INSTRUCTIONS
i
INT
IF
1
<
INSTRUCTIONS
j
INT
IF
1
<
INSTRUCTIONS
k
INT
print
1
ARGS
STR
"test"

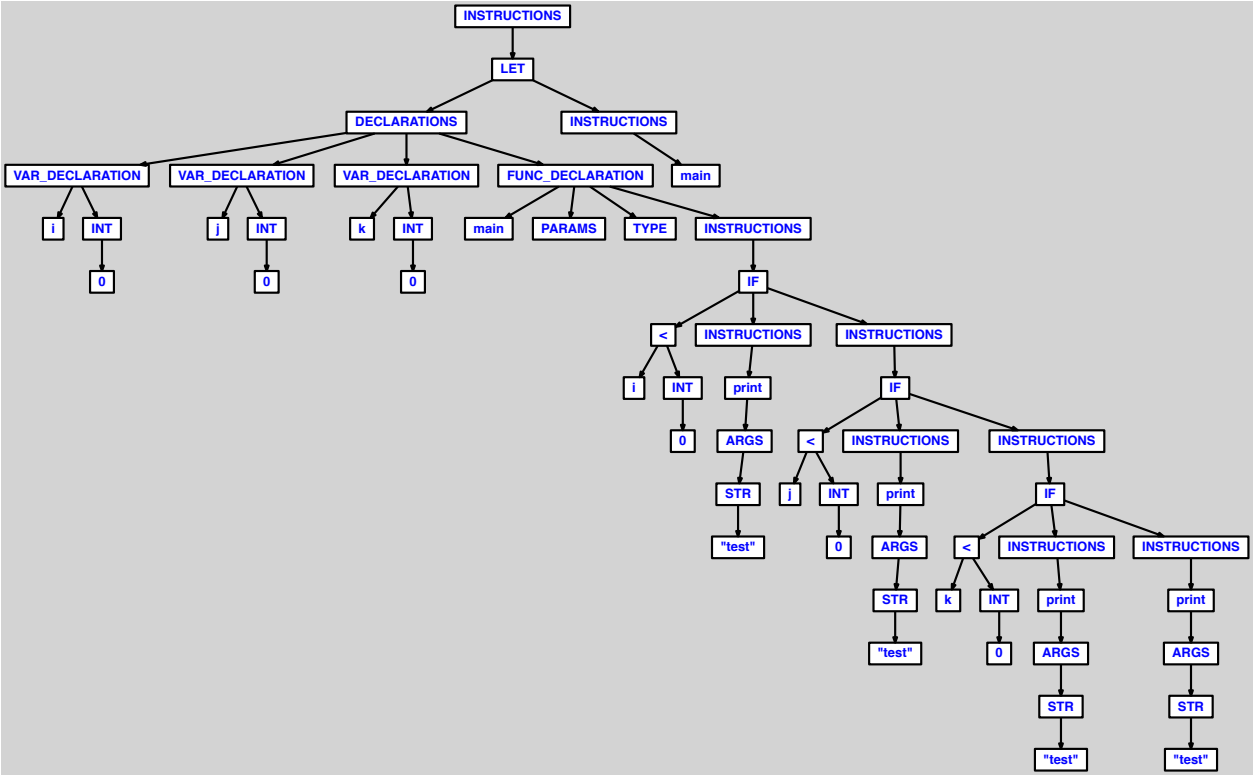
9.2.8 double-imbrication de <if then else>

```
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
main
i
INT
j
INT
main
PARAMS
TYPE
INSTRUCTIONS
0
0
IF
<
INSTRUCTIONS
INSTRUCTIONS
i
INT
print
IF
0
ARGS
<
INSTRUCTIONS
INSTRUCTIONS
STR
j
INT
print
print
"test"
0
ARGS
ARGS
STR
STR
"test"
"test"

9.2.9 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
```

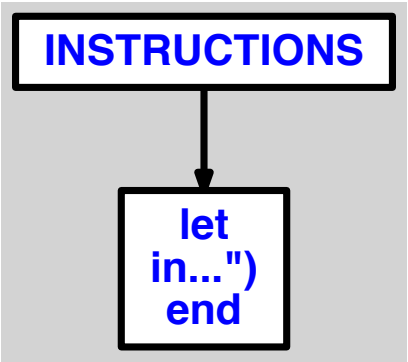

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
0
0
0
IF
<
INSTRUCTIONS
INSTRUCTIONS
i
INT
print
IF
0
ARGS
<
INSTRUCTIONS
INSTRUCTIONS
STR
j
INT
print
IF
"test"
0
ARGS
<
INSTRUCTIONS
INSTRUCTIONS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

10 let

10.1 KO

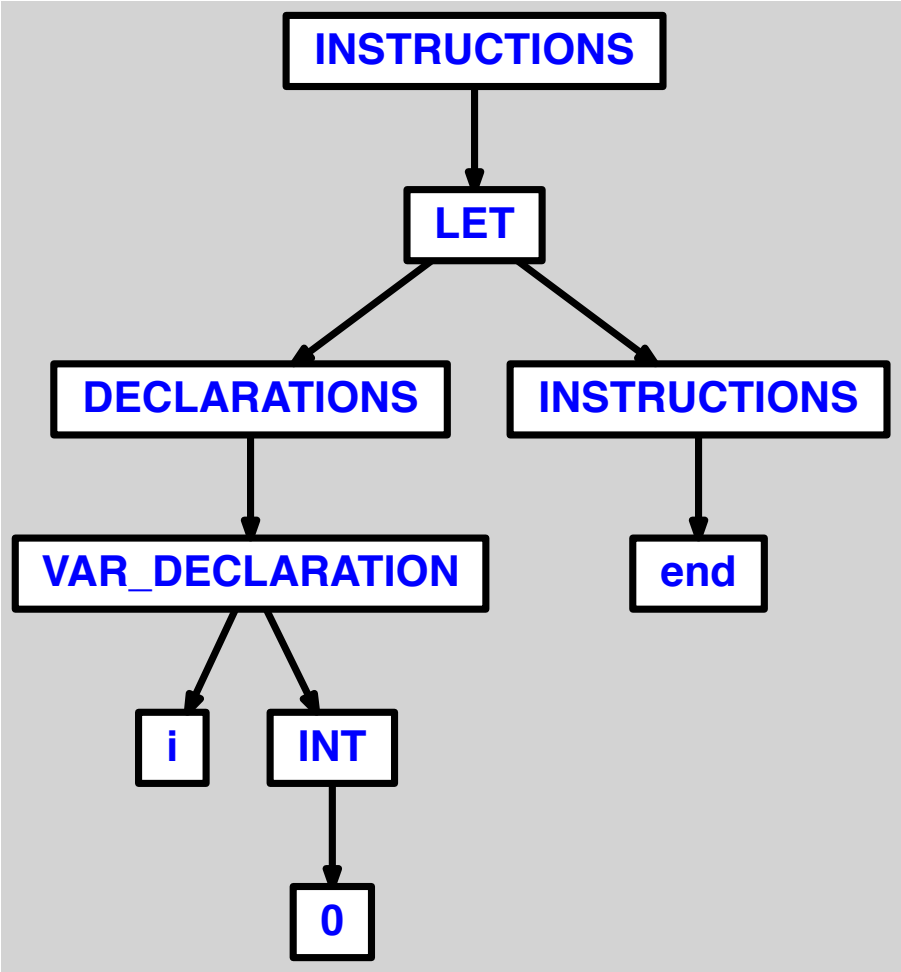
10.1.1 <let> sans declaration

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


INSTRUCTIONS
let
in...")
end

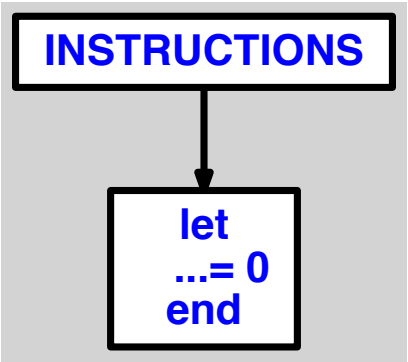
10.1.2 <let> sans instruction

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
end
i
INT
0

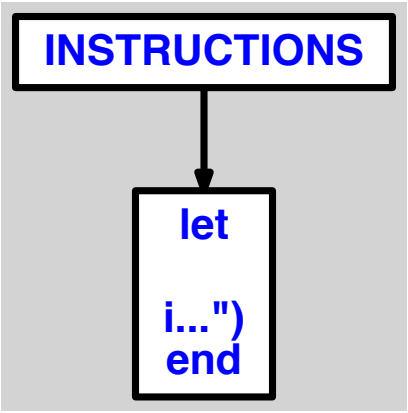
10.1.3 <let> avec inversion de declaration et instruction

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


INSTRUCTIONS
let
...= 0
end

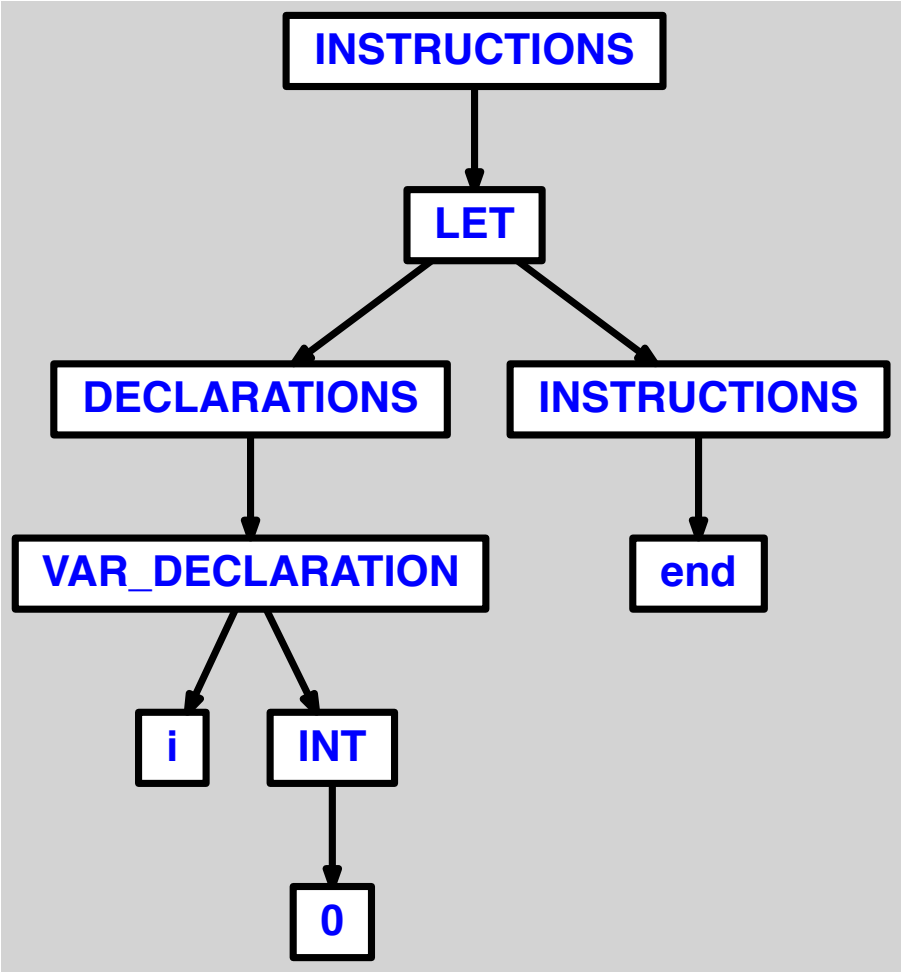
10.1.4 <let> avec declaration vide

```
let

in
  print("test")
end
```


INSTRUCTIONS
let
i...")
end

10.1.5 <let> avec instruction vide

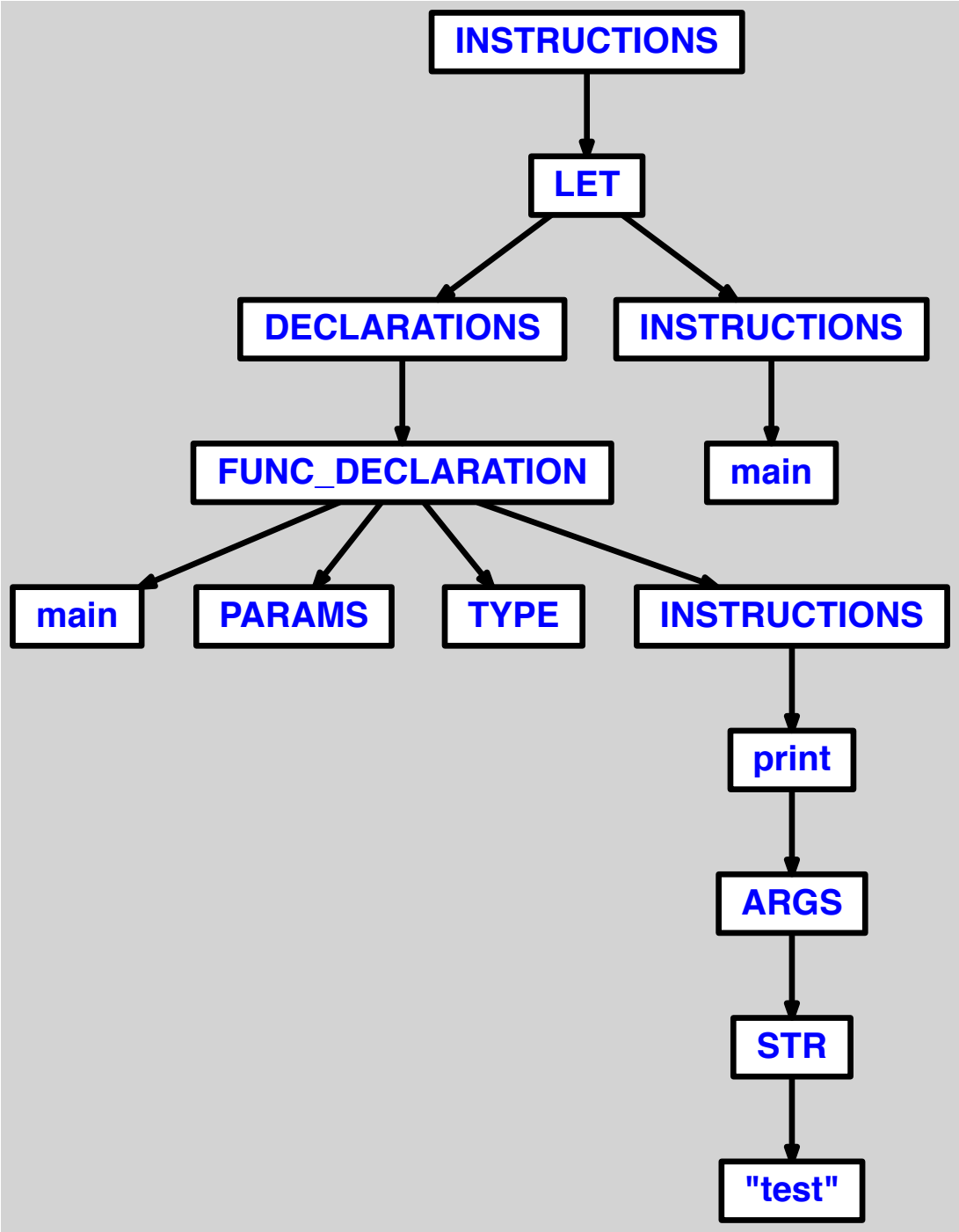
```
let
  var i := 0
in

end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
end
i
INT
0

10.2 OK

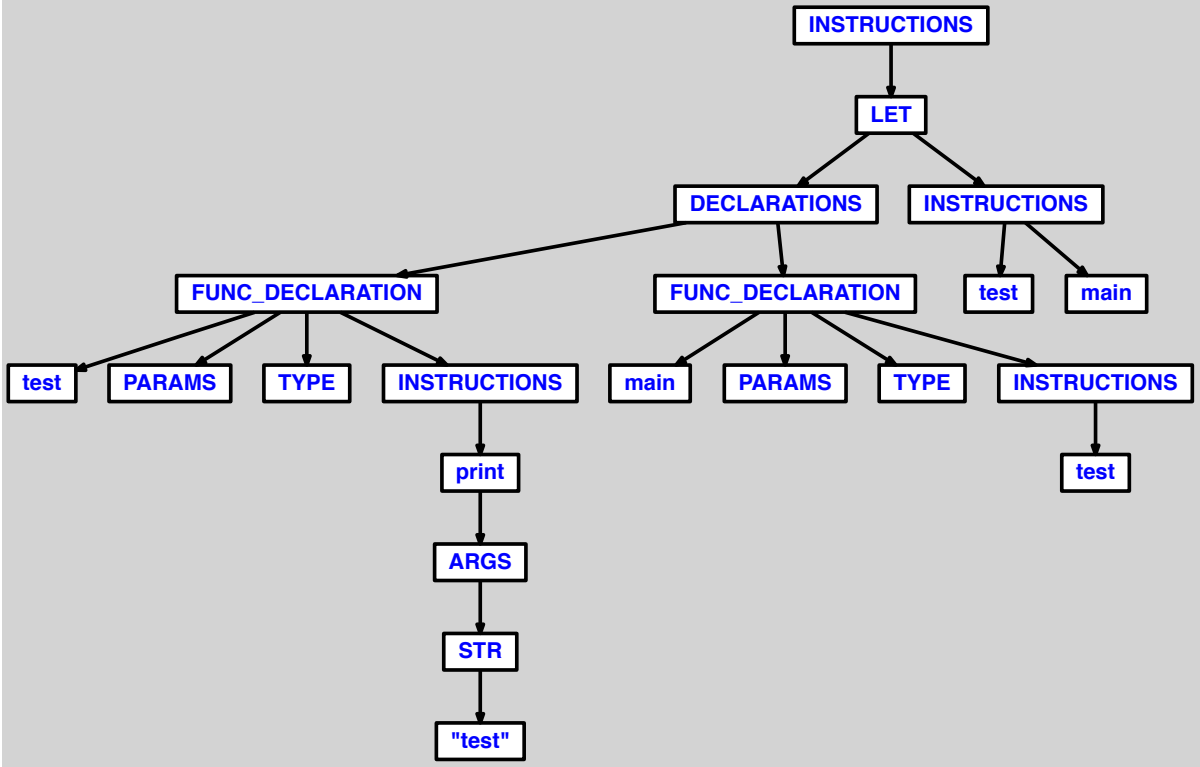
10.2.1 <let> avec 1 declaration de fonction et 1 appel de fonction

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


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

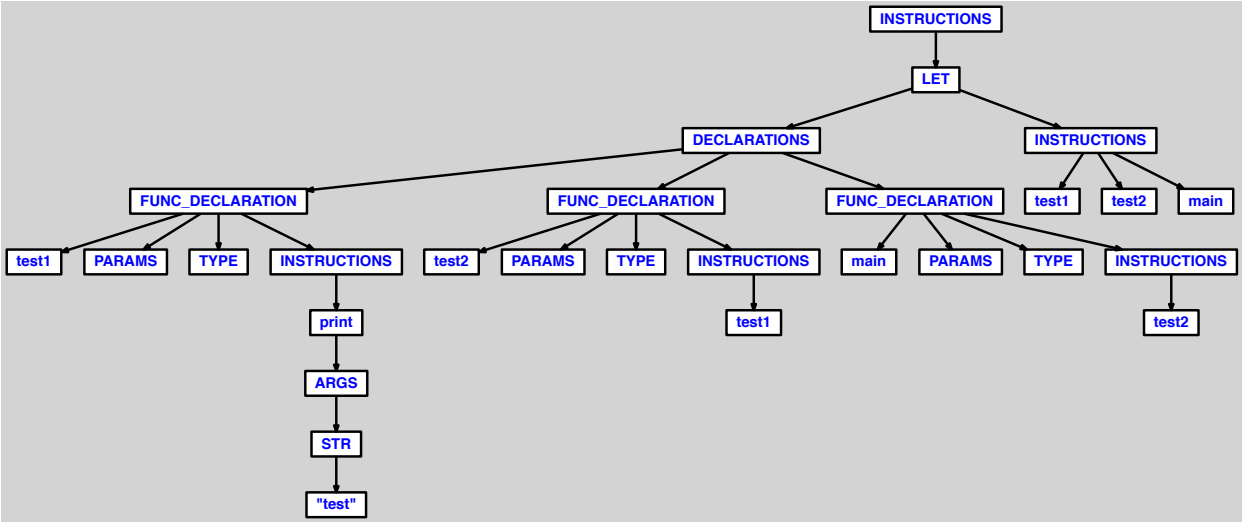
10.2.2 <let> avec 2 declarations de fonction et 2 appels de fonction

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

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


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

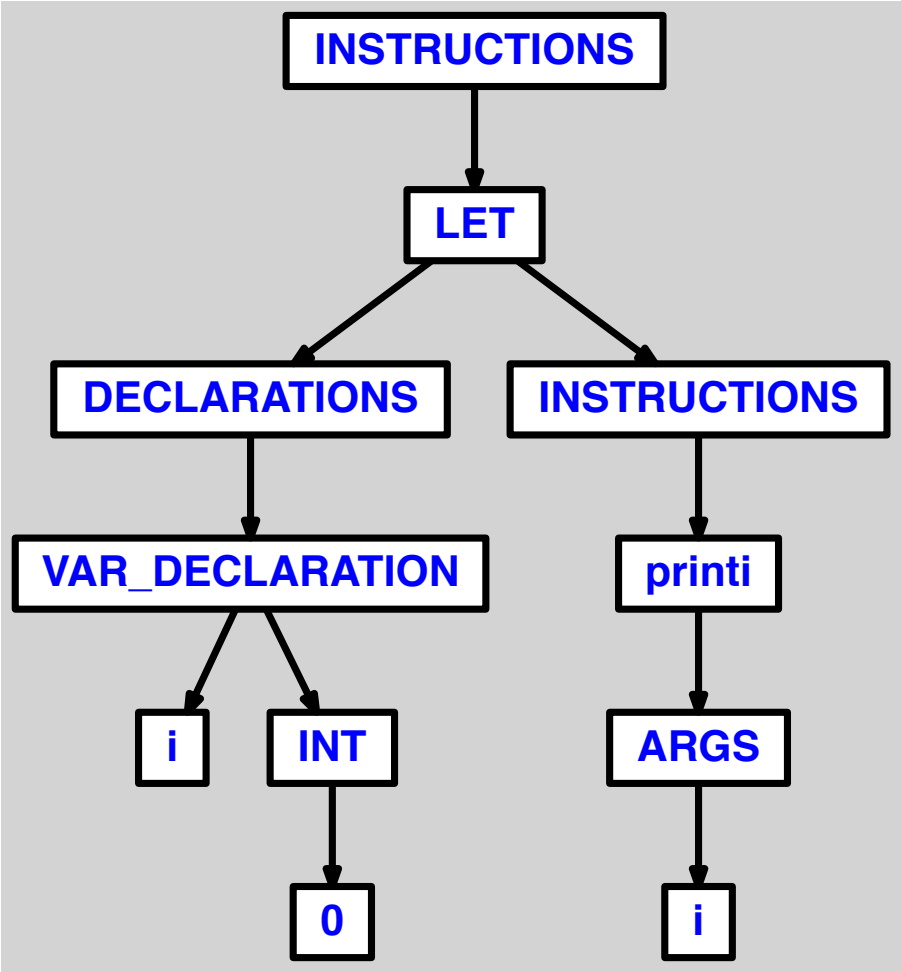
10.2.3 <let> avec 3 declarations de fonction et 3 appels de fonction

```
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
test1
test2
main
test1
PARAMS
TYPE
INSTRUCTIONS
test2
PARAMS
TYPE
INSTRUCTIONS
main
PARAMS
TYPE
INSTRUCTIONS
print
test1
test2
ARGS
STR
"test"

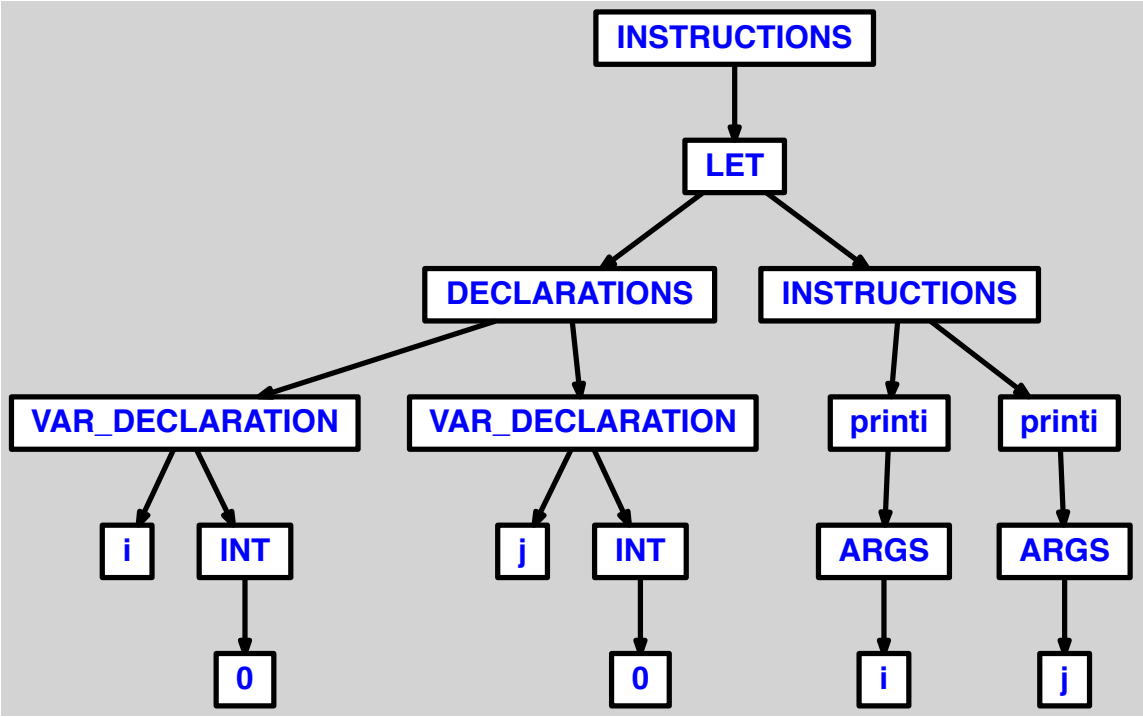
10.2.4 <let> avec 1 declaration de variable et 1 instruction

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
printi
i
INT
ARGS
0
i

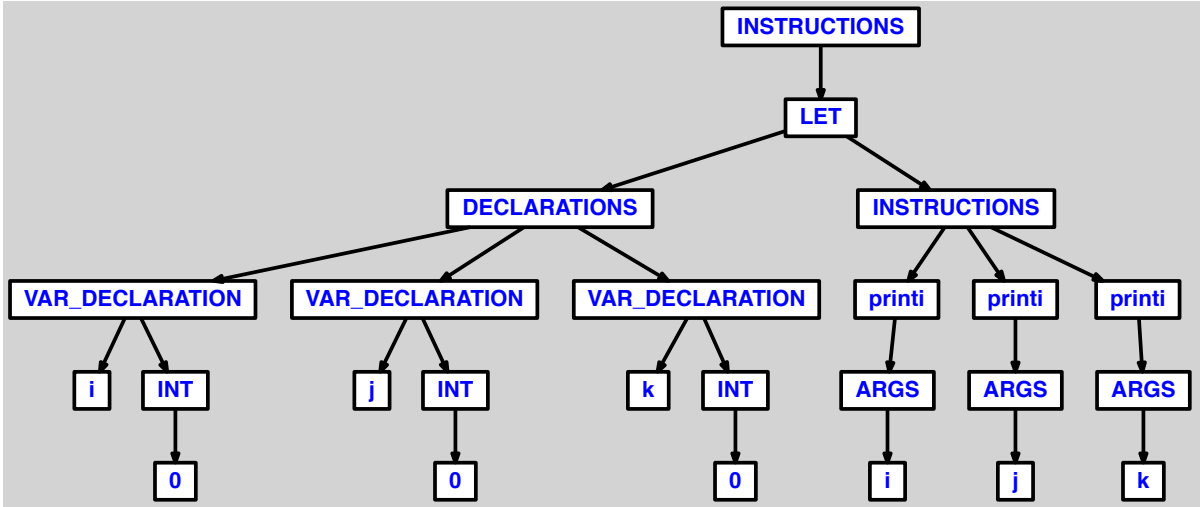
10.2.5 <let> avec 2 declarations de variable et 2 instructions

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


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

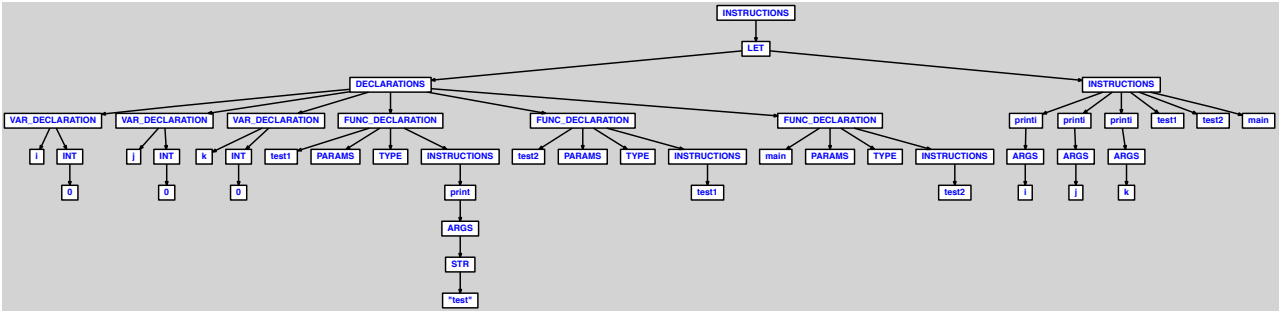
10.2.6 <let> avec 3 declarations de variable et 3 instructions

```
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
printi
printi
printi
i
INT
j
INT
k
INT
ARGS
ARGS
ARGS
0
0
0
i
j
k

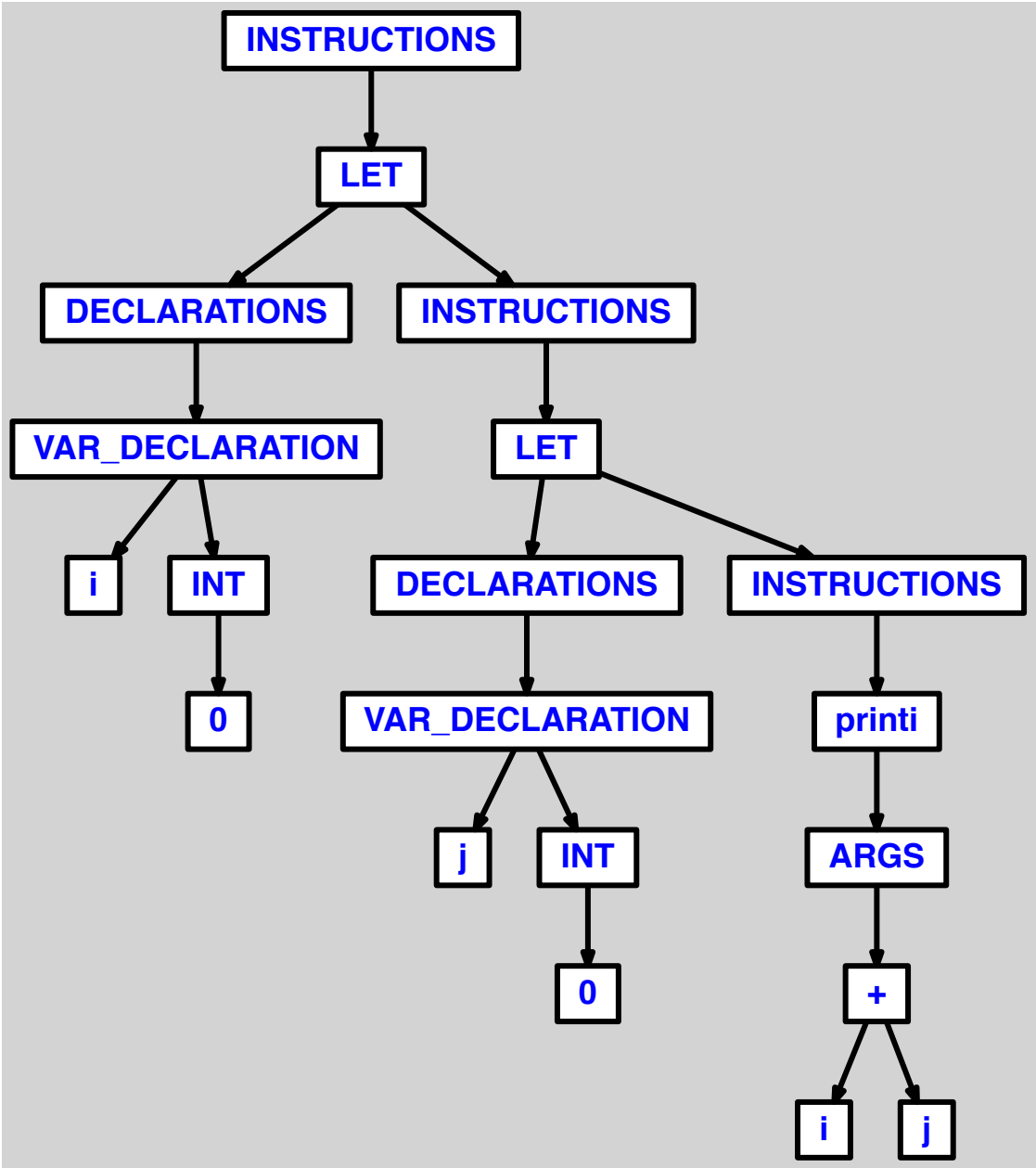
10.2.7 <let> avec 3 declarations de variable et fonction, et 3 instructions et appels de fonction

```
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
printl
printl
printl
test1
test2
main
i
INT
j
INT
k
INT
test1
PARAMS
TYPE
INSTRUCTIONS
test2
PARAMS
TYPE
INSTRUCTIONS
main
PARAMS
TYPE
INSTRUCTIONS
ARGS
ARGS
ARGS
0
0
0
print
test1
test2
i
j
k
ARGS
STR
"test"

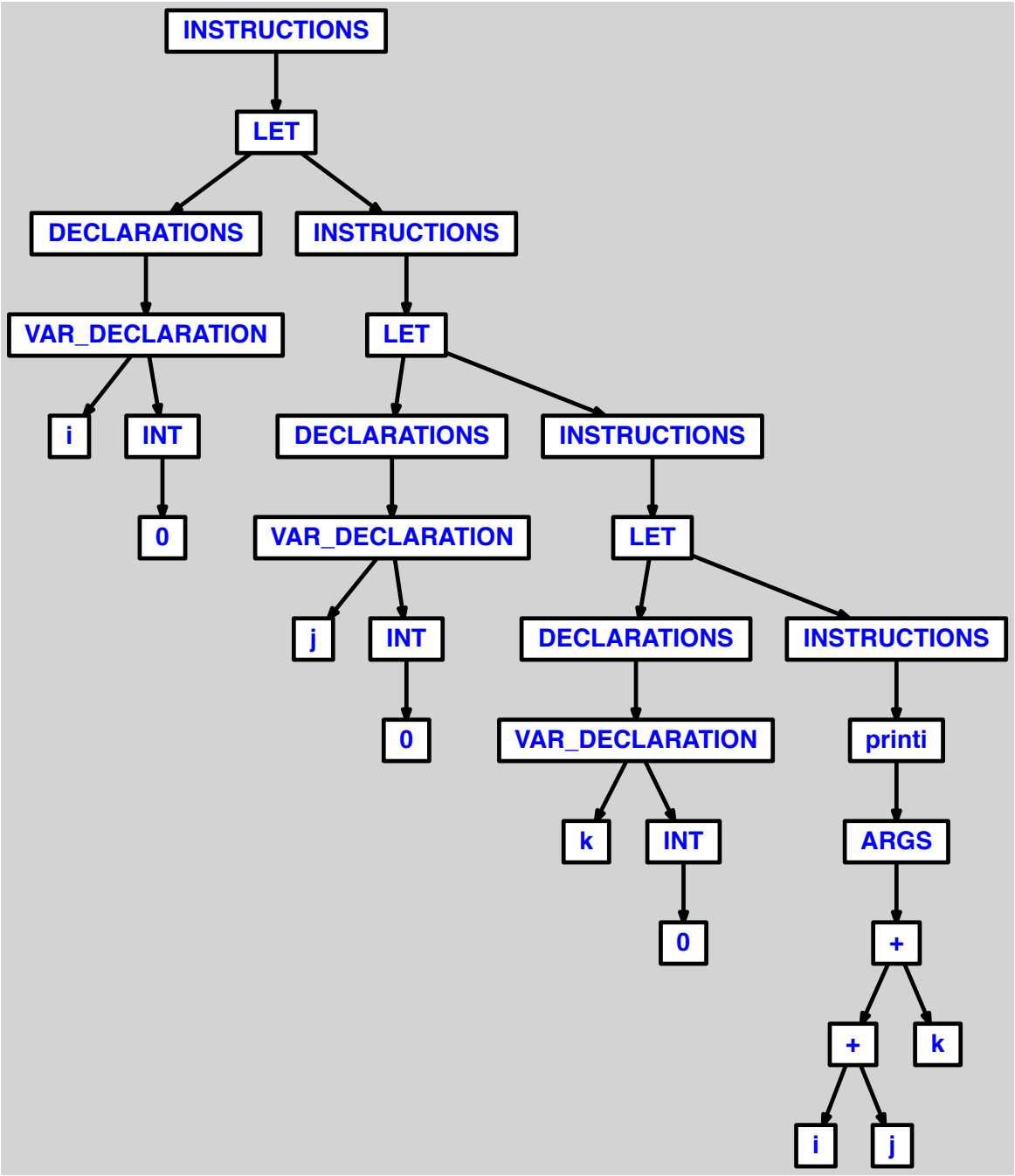
10.2.8 <let> avec double-imbrication

```
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
printi
j
INT
ARGS
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
```

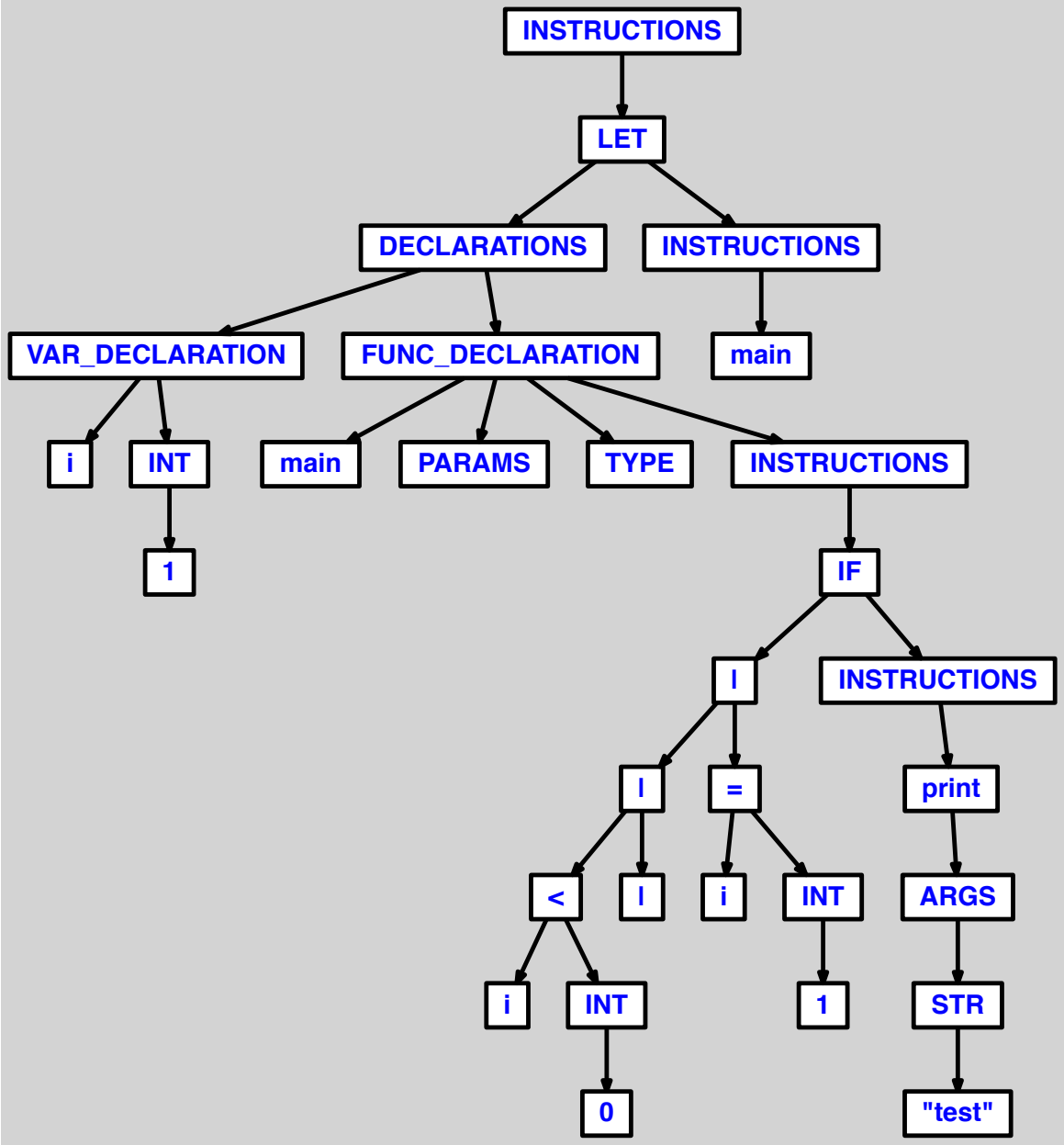

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
LET
i
INT
DECLARATIONS
INSTRUCTIONS
0
VAR_DECLARATION
LET
j
INT
DECLARATIONS
INSTRUCTIONS
0
VAR_DECLARATION
printi
k
INT
ARGS
0
+
+
k
i
j

11 or

11.1 KO

11.1.1 <|> mal ecrit

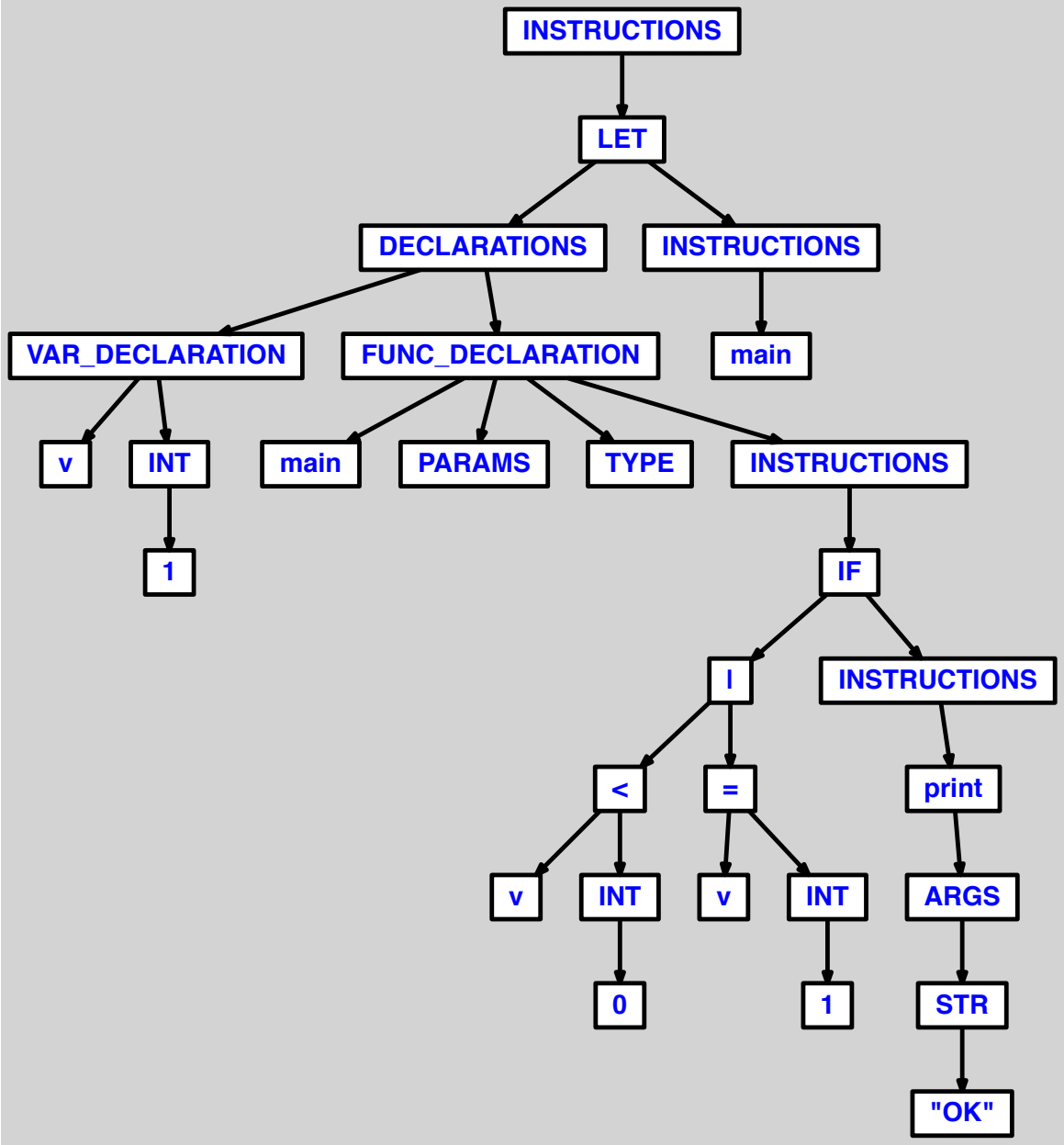
```
let
  var i := 1

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


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

11.2 OK

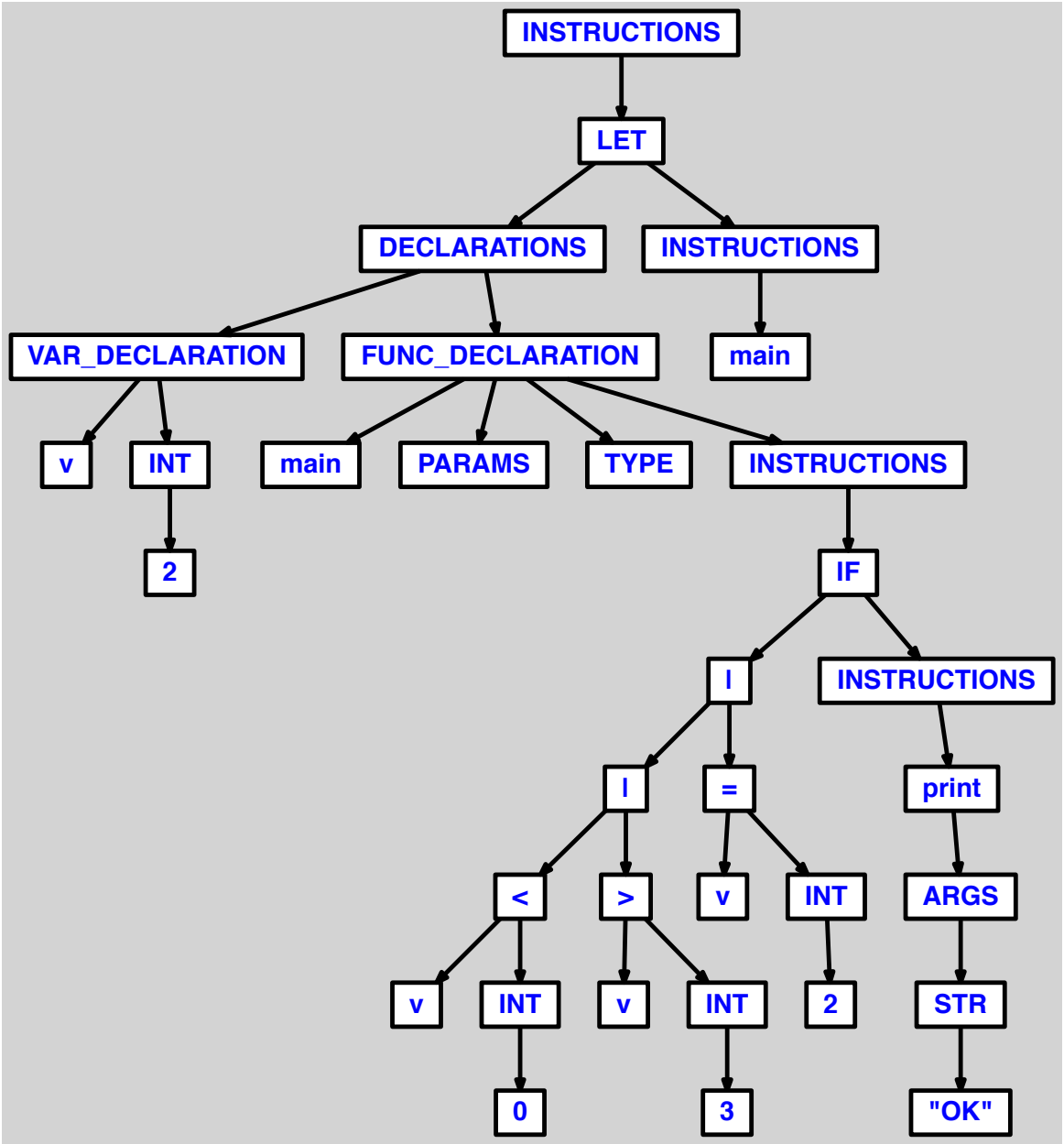
11.2.1 condition avec 1 <|>

```
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
main
v
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
IF
|
INSTRUCTIONS
<
=
print
v
INT
v
INT
ARGS
0
1
STR
"OK"

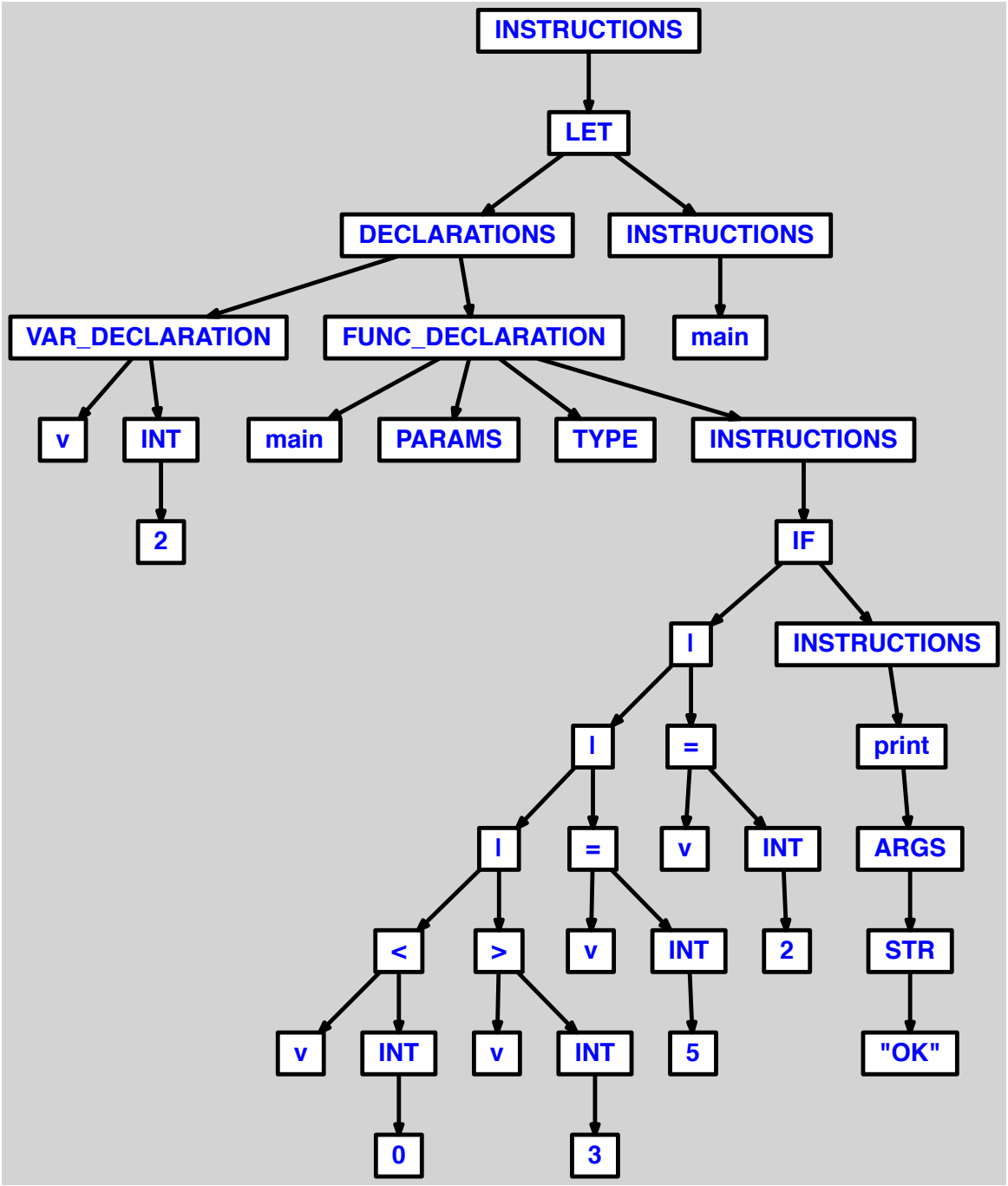
11.2.2 condition avec 2 <|>

```
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
main
v
INT
main
PARAMS
TYPE
INSTRUCTIONS
2
IF
|
INSTRUCTIONS
|
=
print
<
>
v
INT
ARGS
v
INT
v
INT
2
STR
0
3
"OK"

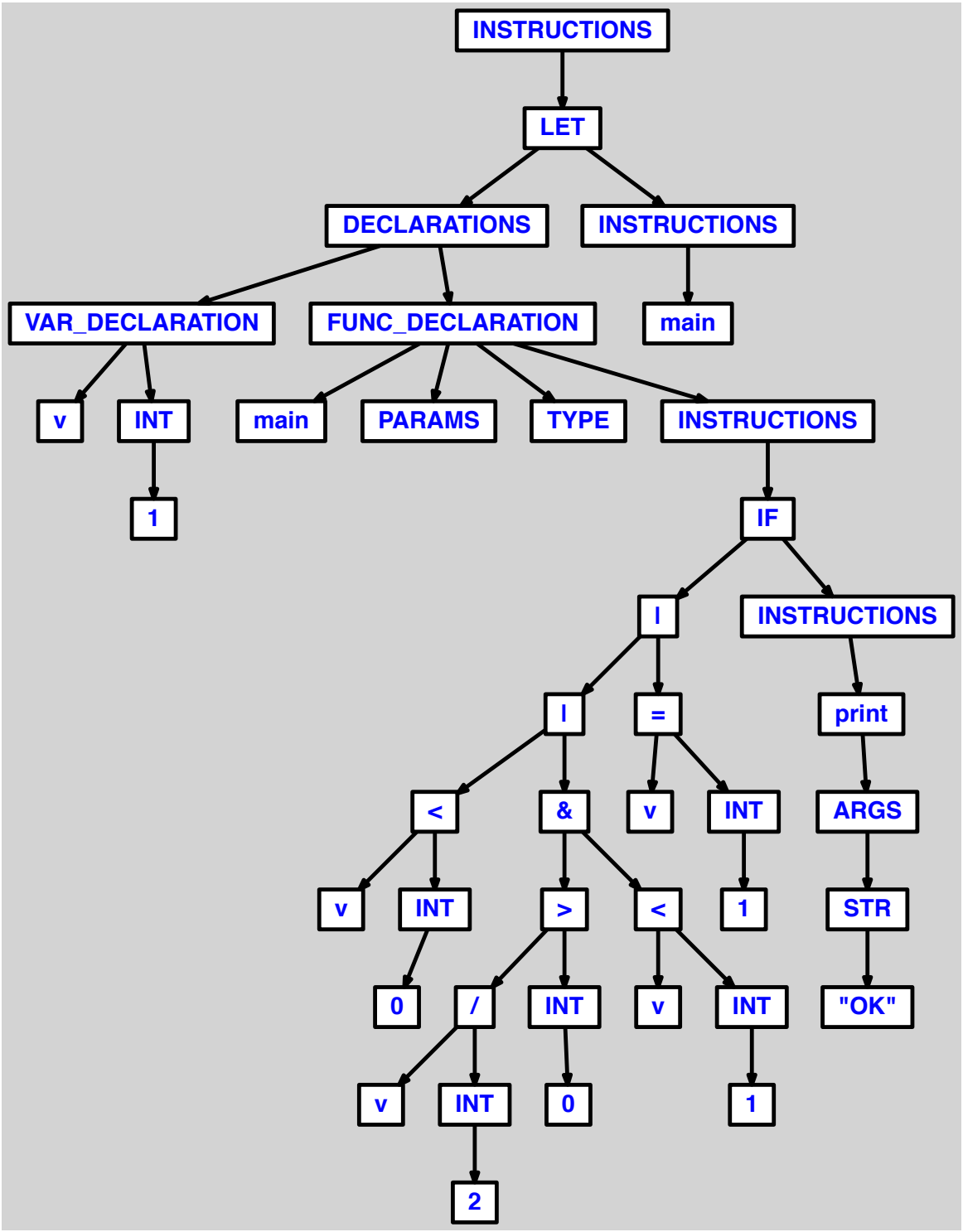
11.2.3 condition avec 3 <|>

```
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
main
v
INT
main
PARAMS
TYPE
INSTRUCTIONS
2
IF
I
INSTRUCTIONS
I
=
print
I
=
v
INT
ARGS
<
>
v
INT
2
STR
v
INT
v
INT
5
"OK"
0
3

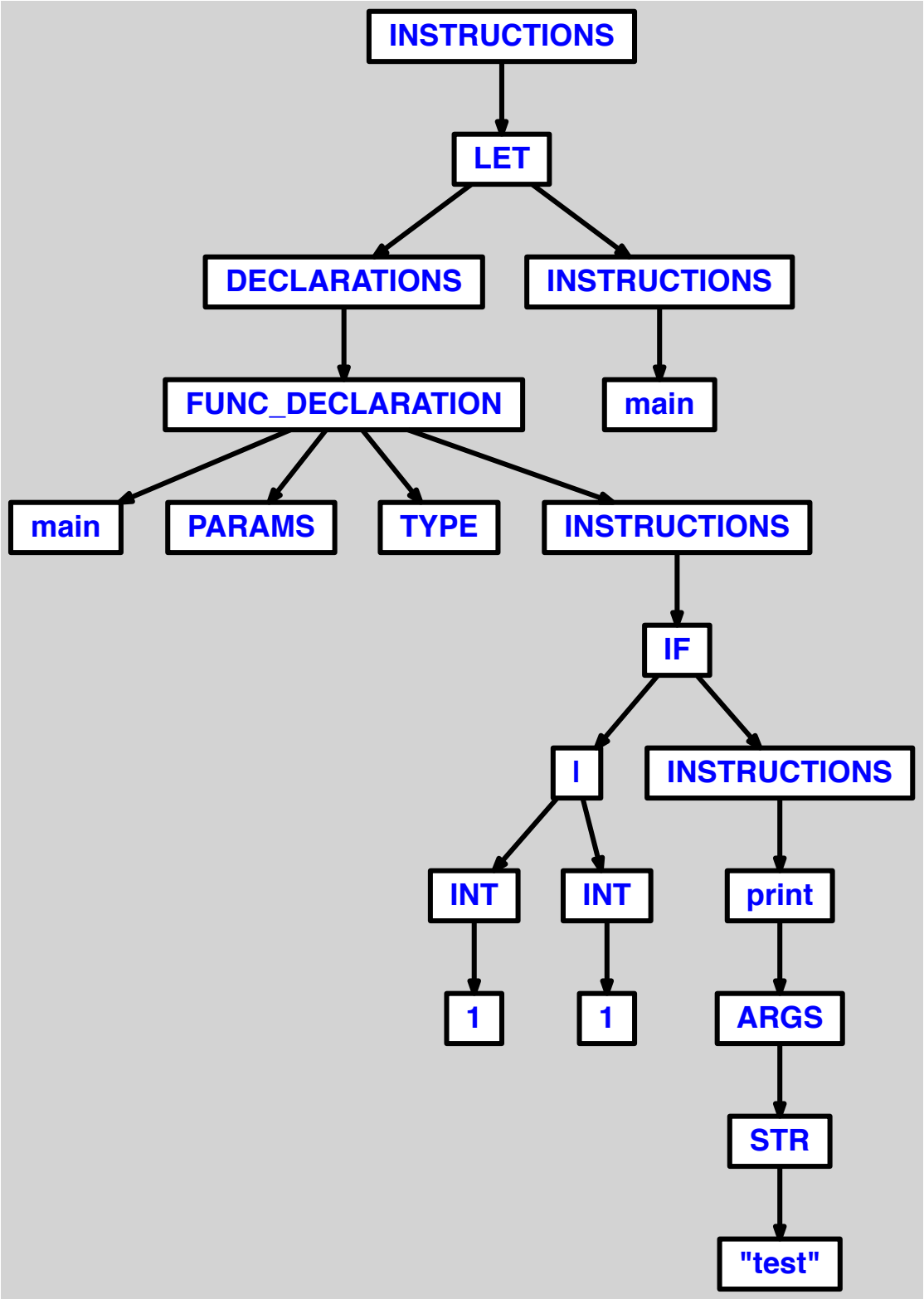
11.2.4 condition avec 2 <|> et 1 <&>

```
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
main
v
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
IF
|
INSTRUCTIONS
|
=
print
<
&
v
INT
ARGS
v
INT
>
<
1
STR
0
/
INT
v
INT
"OK"
v
INT
0
1
2

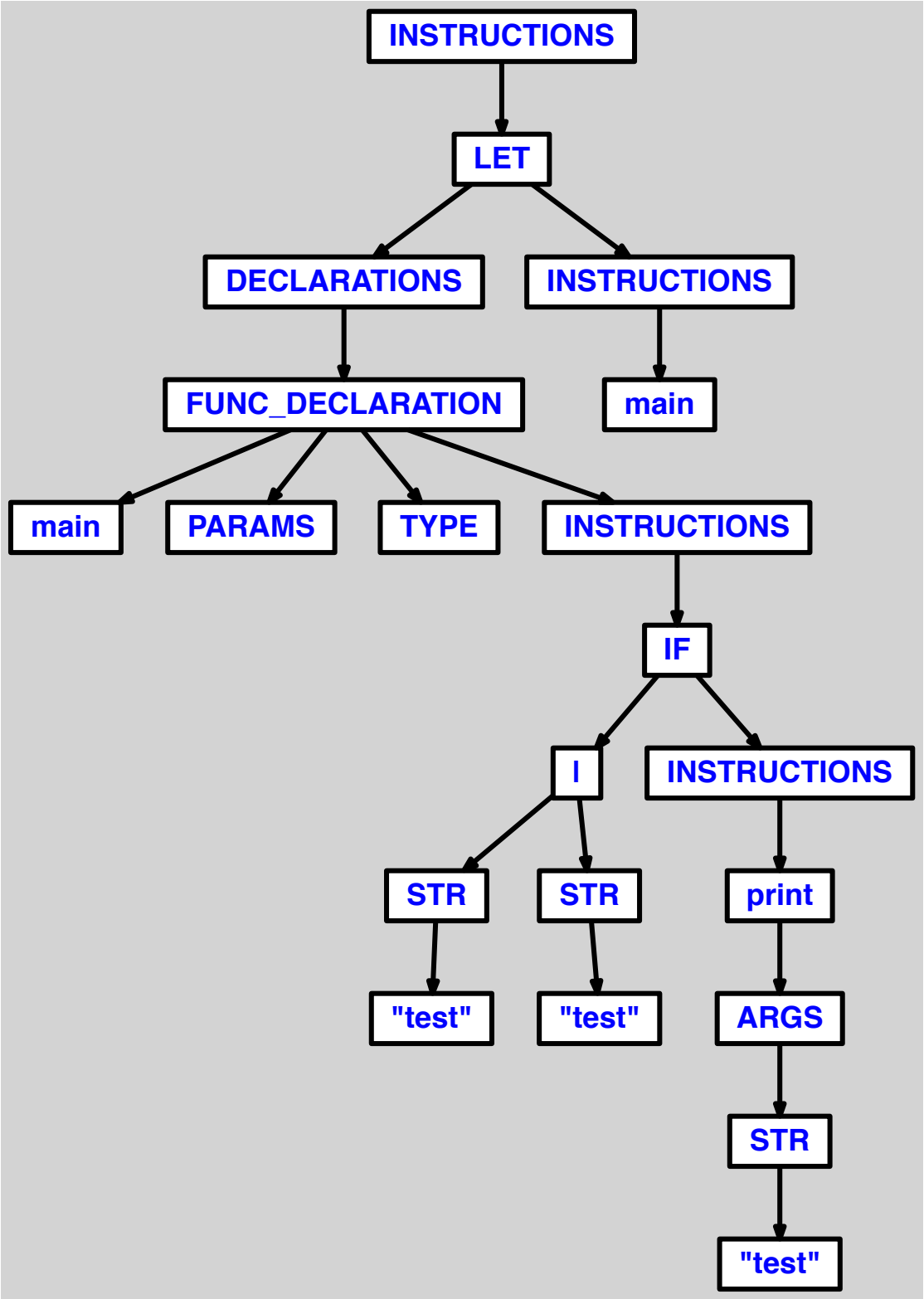
11.2.5 <|> avec des entiers

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


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

11.2.6 <|> avec des chaines

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

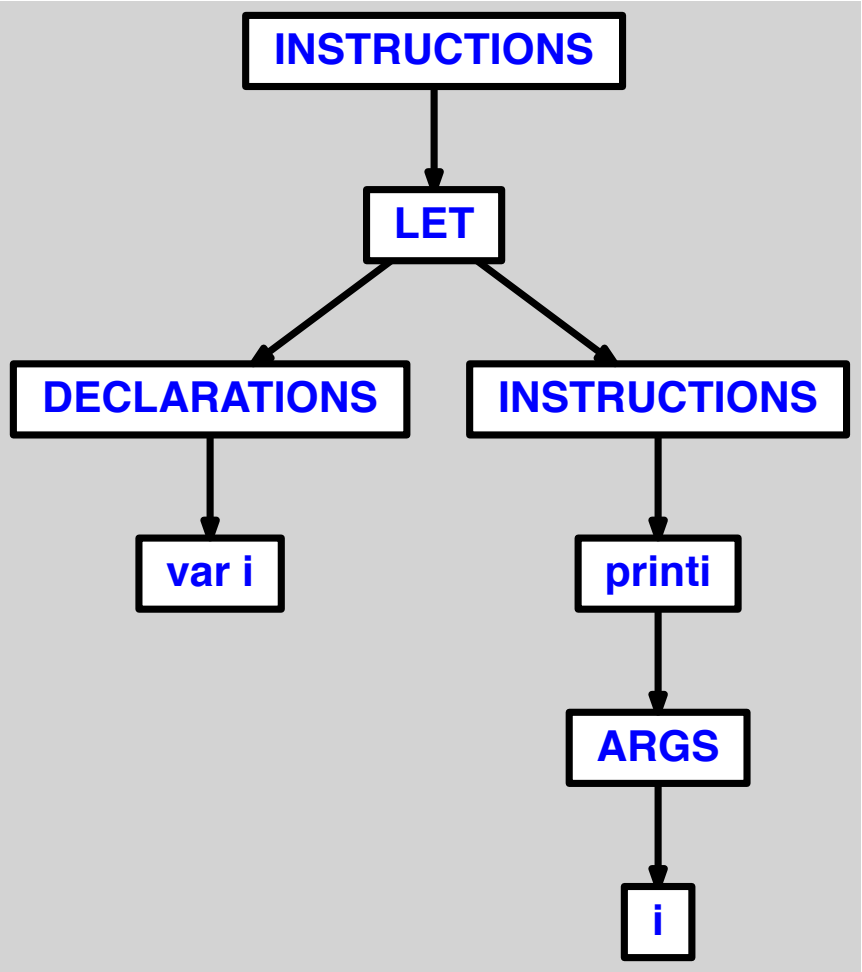

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
IF
I
INSTRUCTIONS
STR
STR
print
"test"
"test"
ARGS
STR
"test"

12 var

12.1 KO

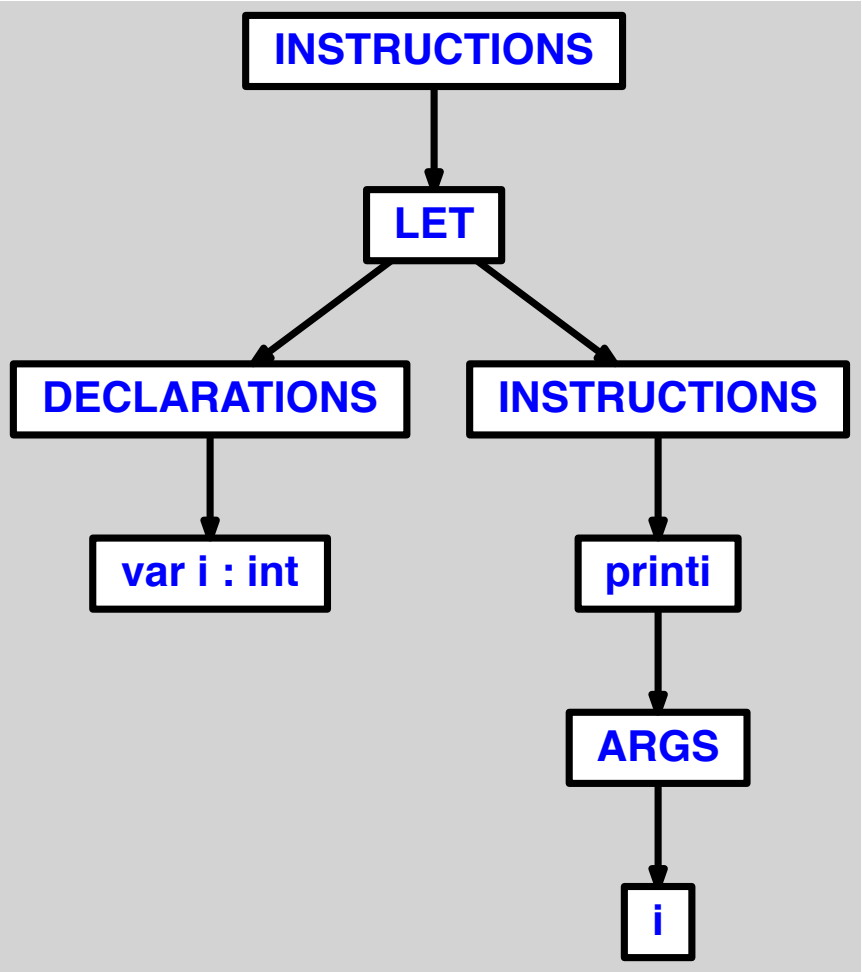
12.1.1 declaration sans affectation

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
var i
printi
ARGS
i

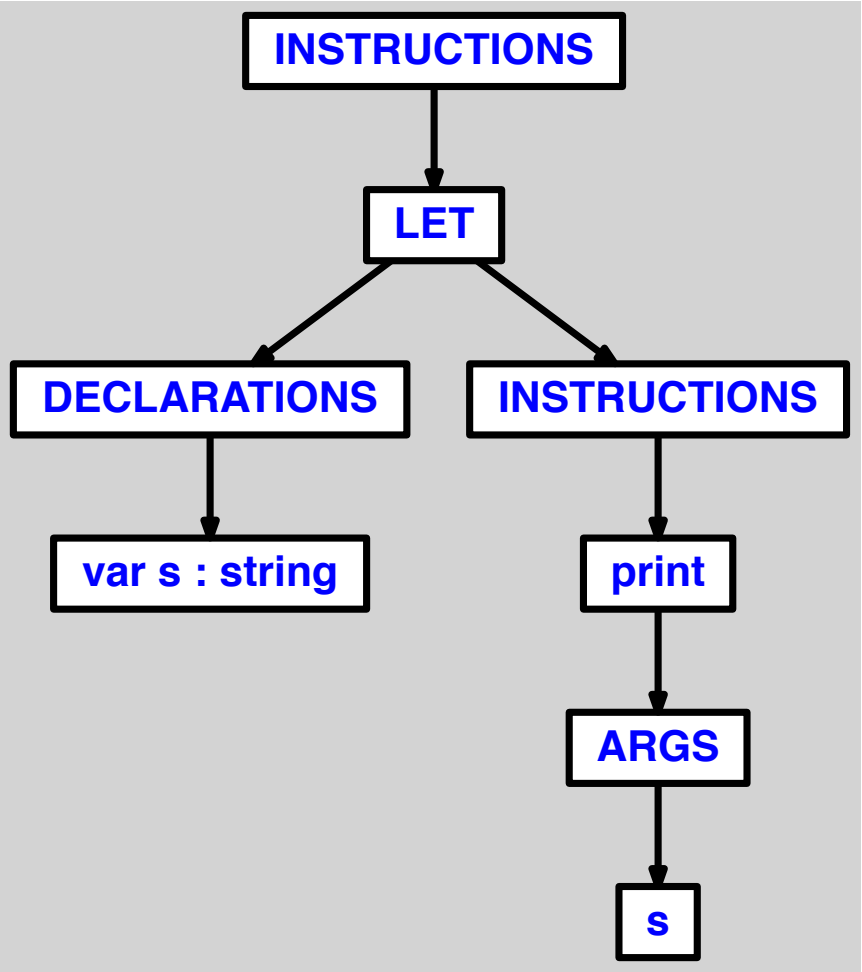
12.1.2 declaration d'entier avec type sans affectation

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
var i : int
printi
ARGS
i

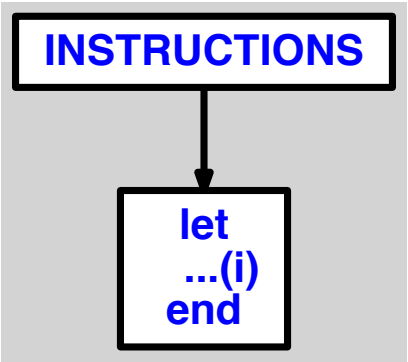
12.1.3 declaration de chaine avec type sans affectation

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
var s : string
print
ARGS
s

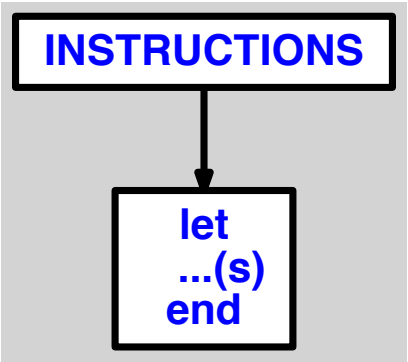
12.1.4 affectation d'entier sans declaration

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


INSTRUCTIONS
let
...(i)
end

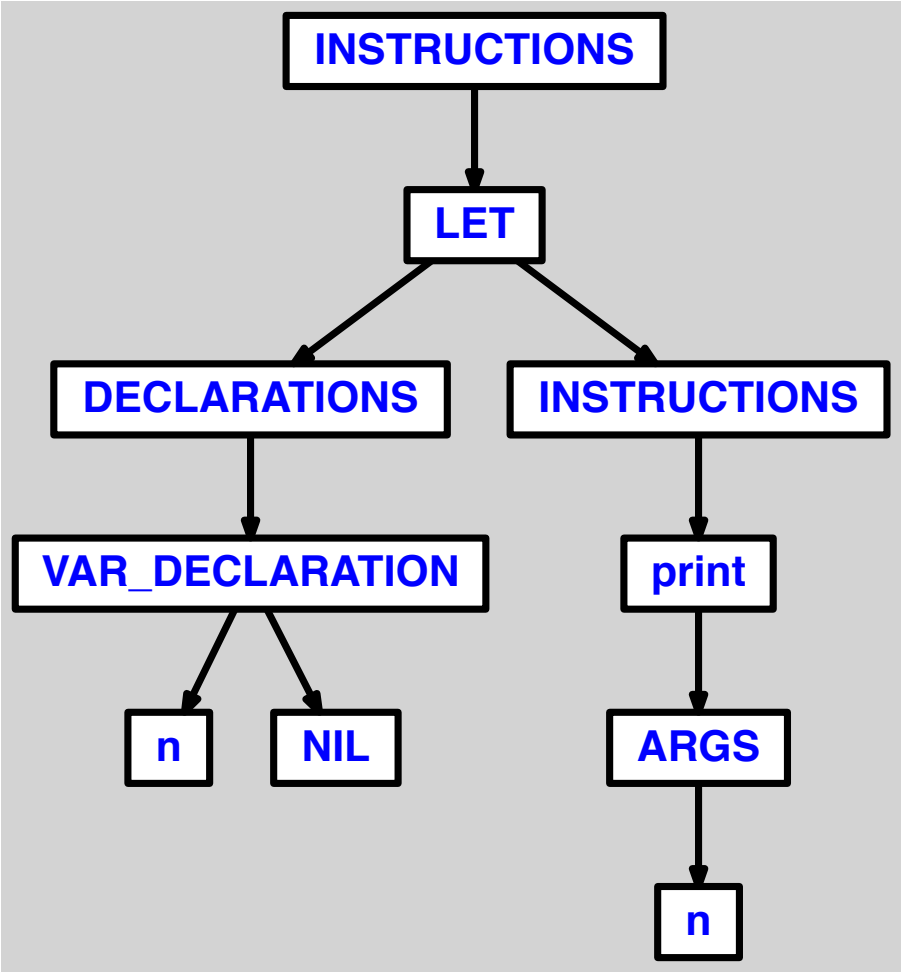
12.1.5 affectation de chaine sans declaration

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


INSTRUCTIONS
let
...(s)
end

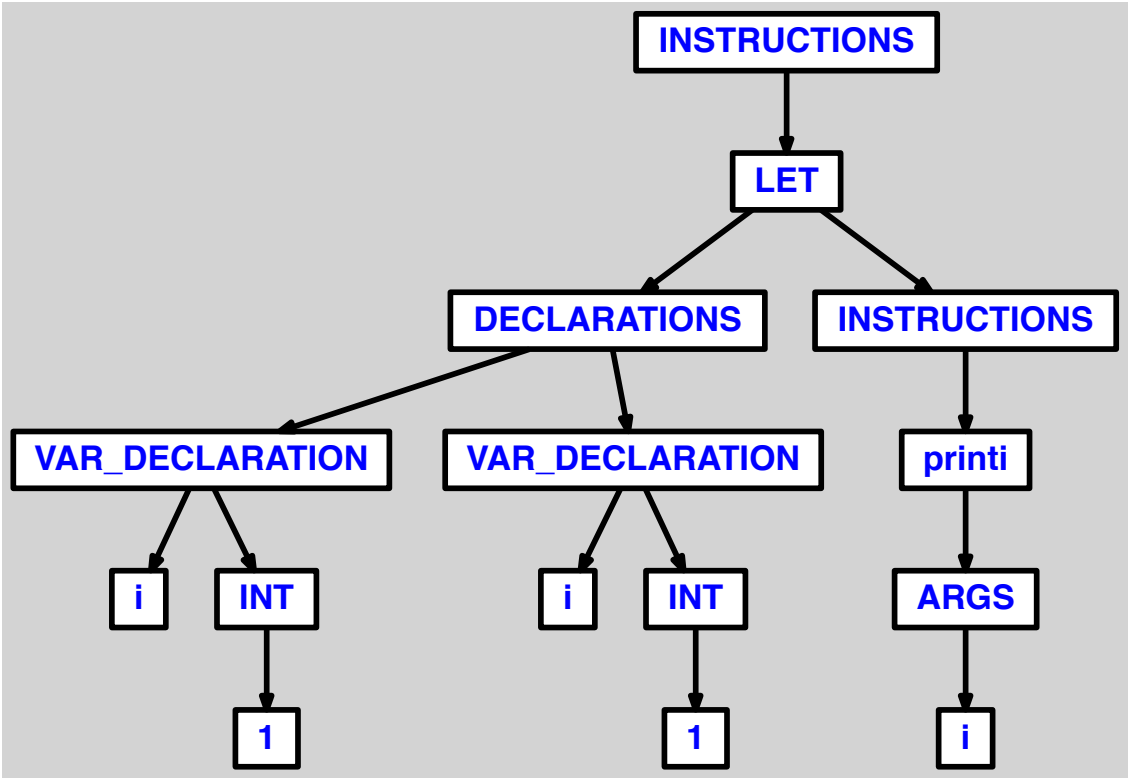
12.1.6 affectation de <nil>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
print
n
NIL
ARGS
n

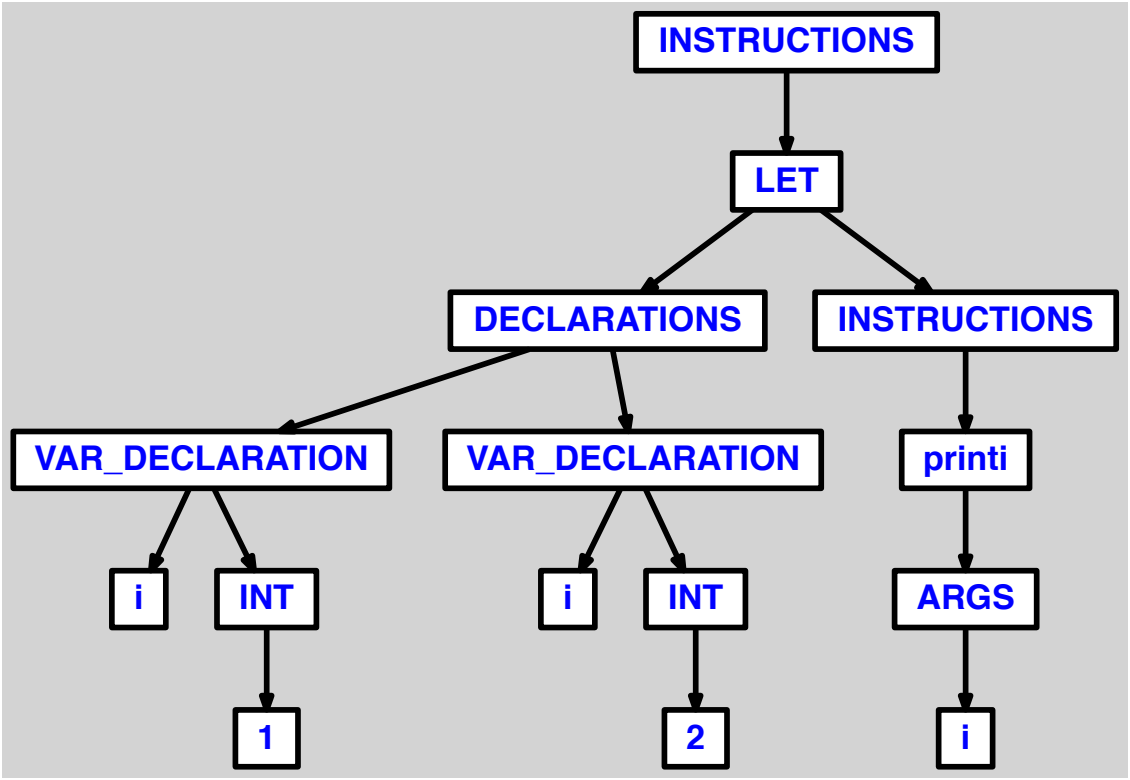
12.1.7 double-declaration avec valeurs egales

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


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

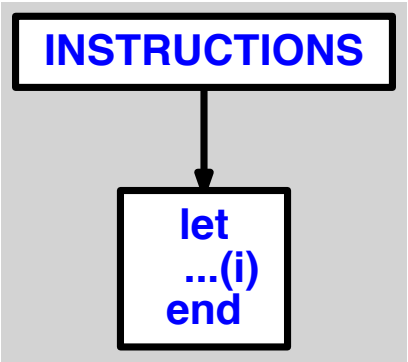
12.1.8 double-declaration avec valeurs differentes

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


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

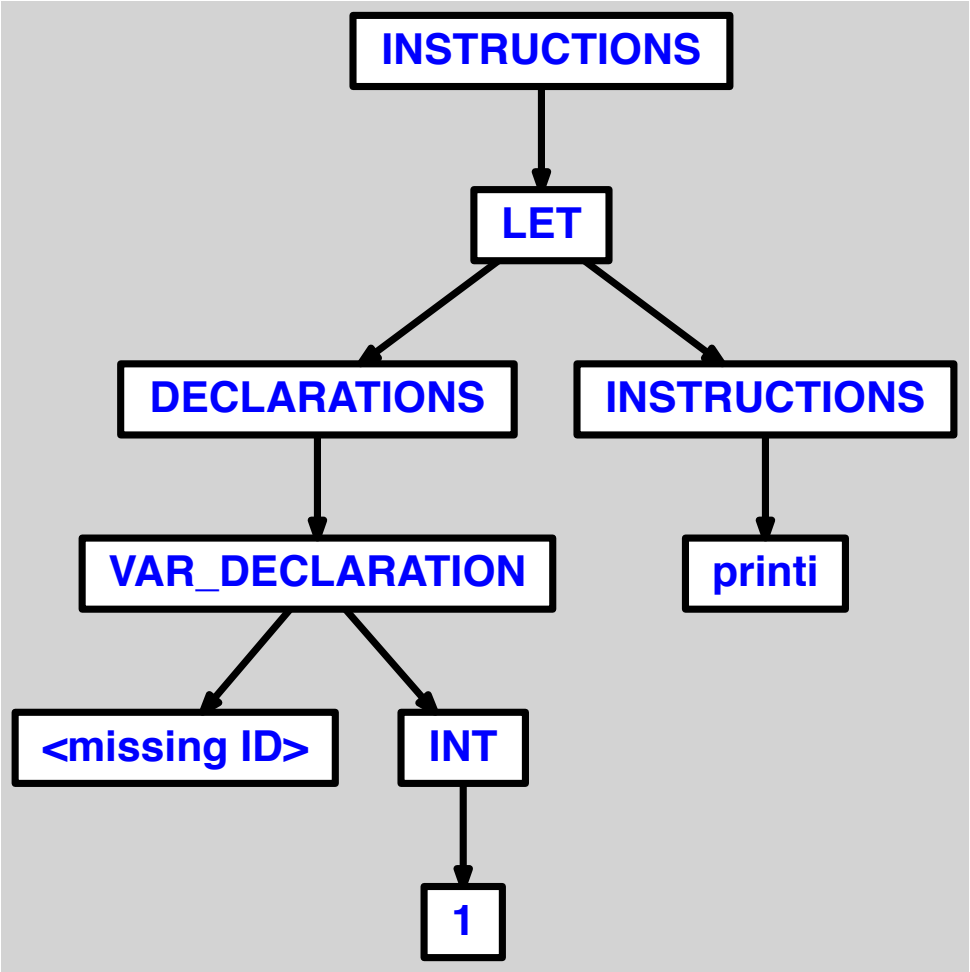
12.1.9 <var> mal ecrit

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


INSTRUCTIONS
let
...(i)
end

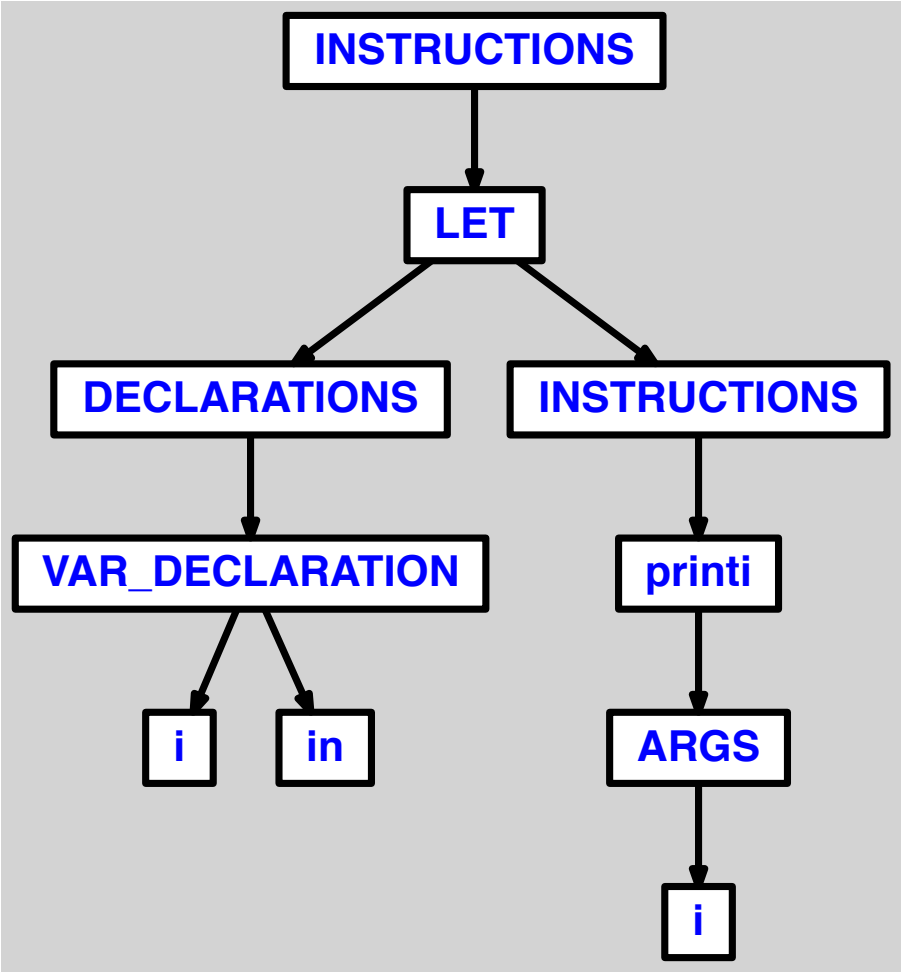
12.1.10 declaration de caractere special

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


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

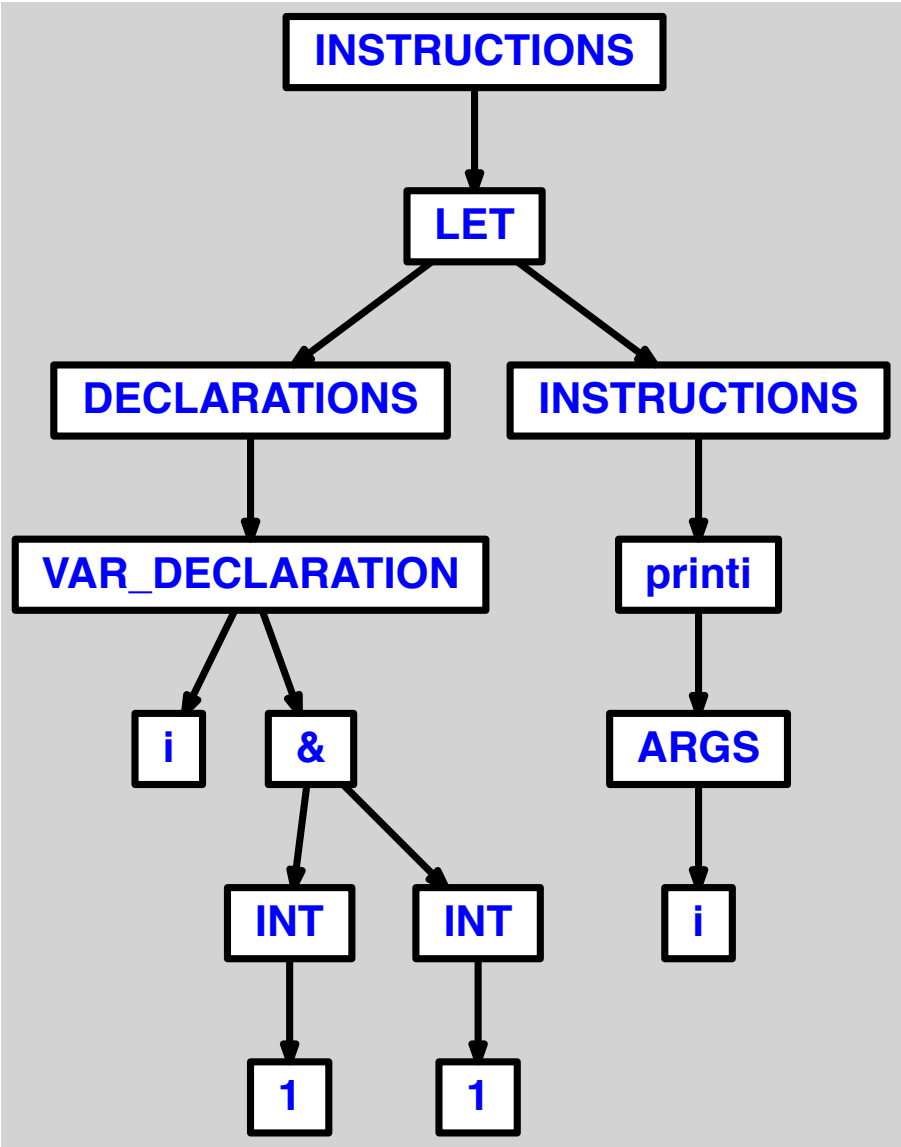
12.1.11 affectation de caractere special

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
printi
i
in
ARGS
i

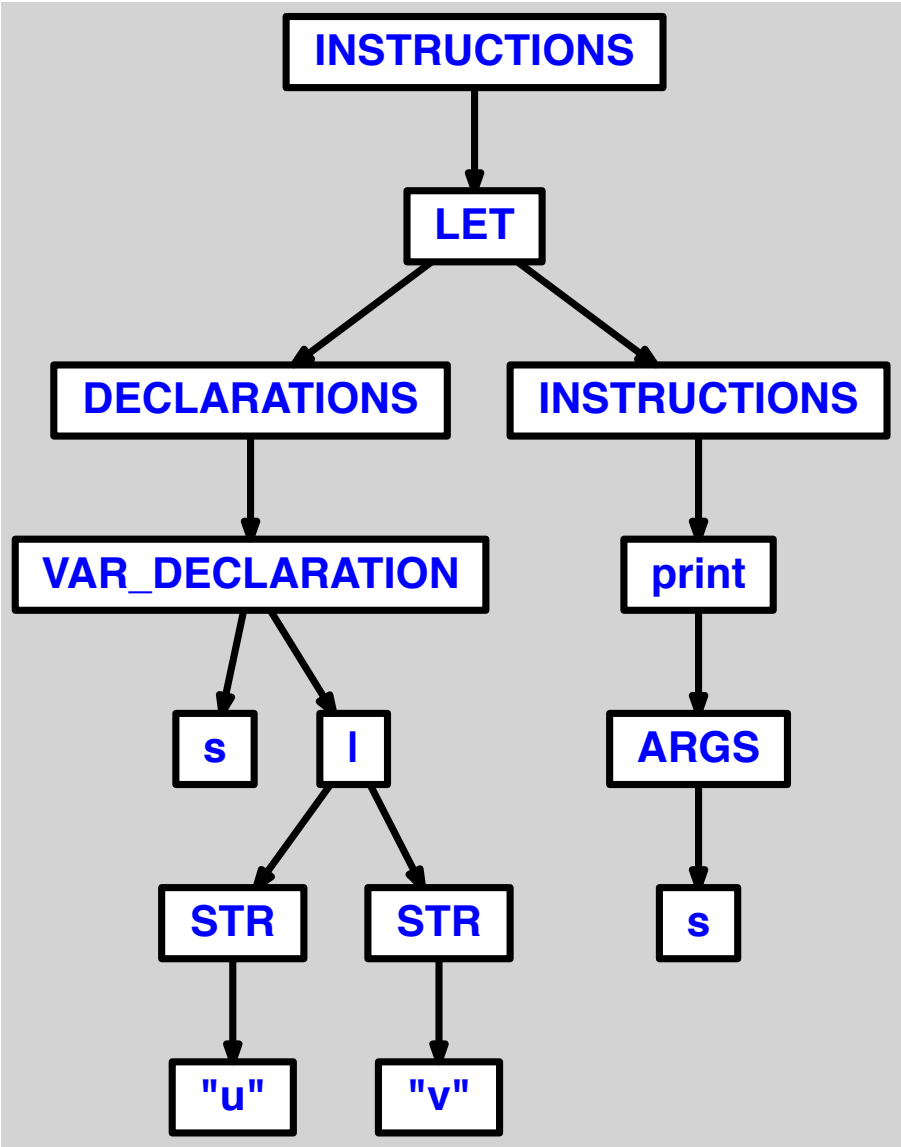
12.1.12 affectation d'operation logique sur entiers

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


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

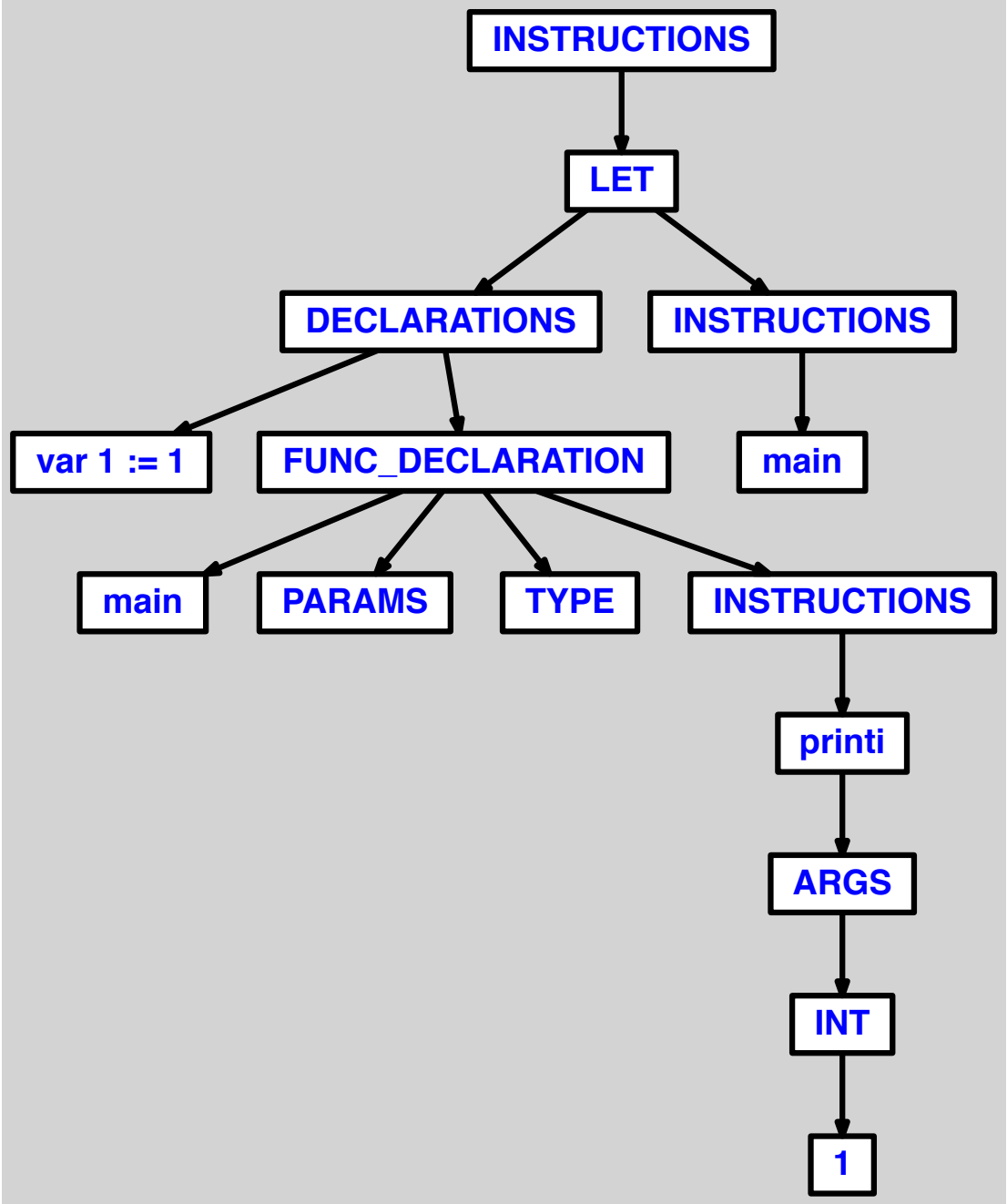
12.1.13 affectation d'operation logique sur chaines

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


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

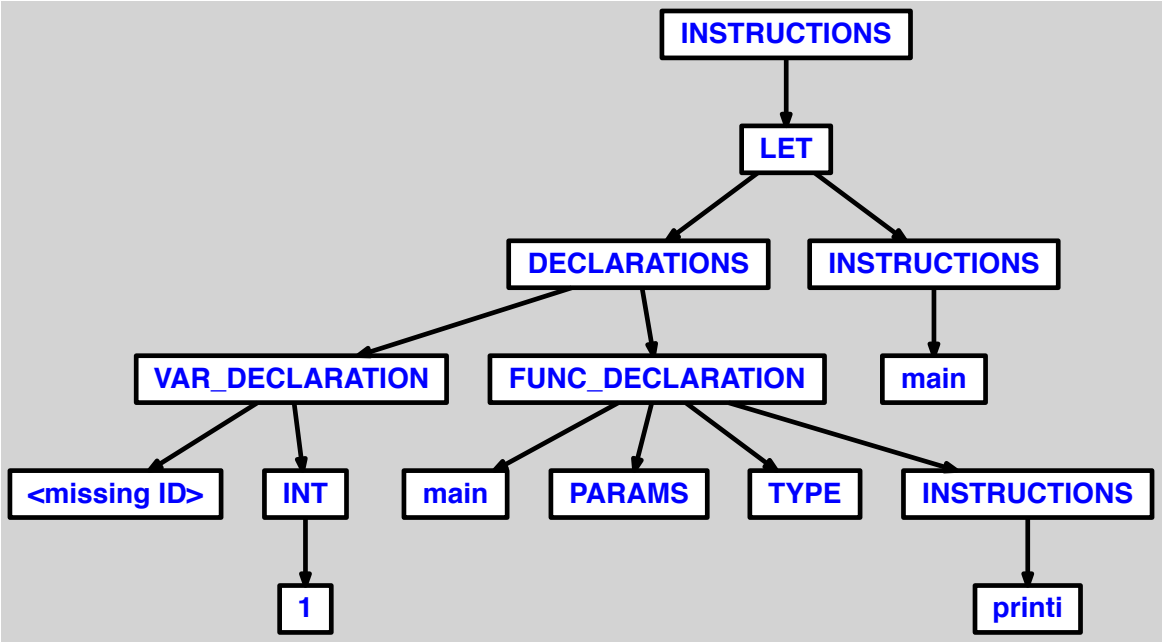
12.1.14 chiffre comme nom de variable

```
let
  var 1 := 1

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


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

12.1.15 <_> comme nom de variable

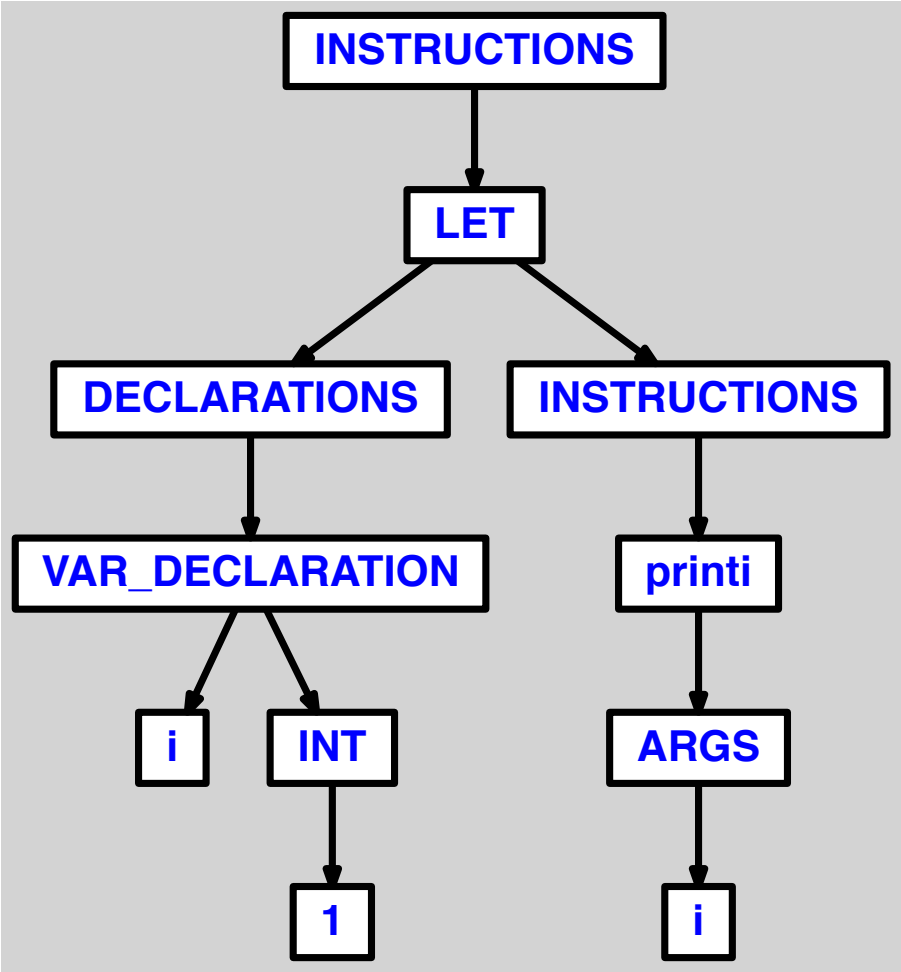
```
let
  var _ := 1

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


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

12.2 OK

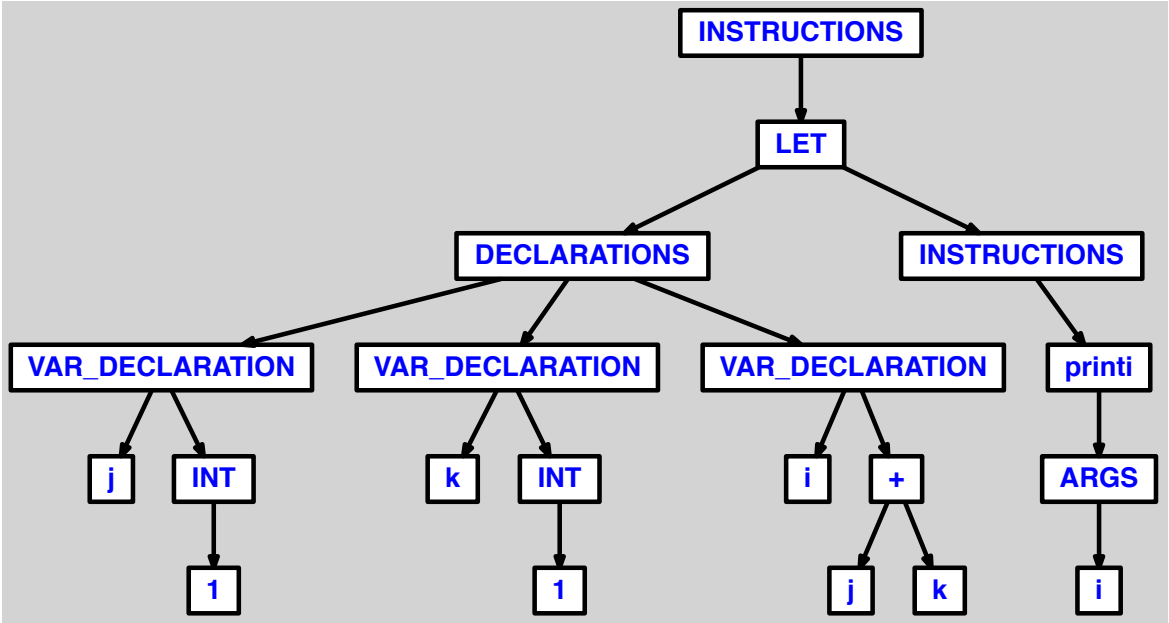
12.2.1 declaration d'entier simple

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
printi
i
INT
ARGS
1
i

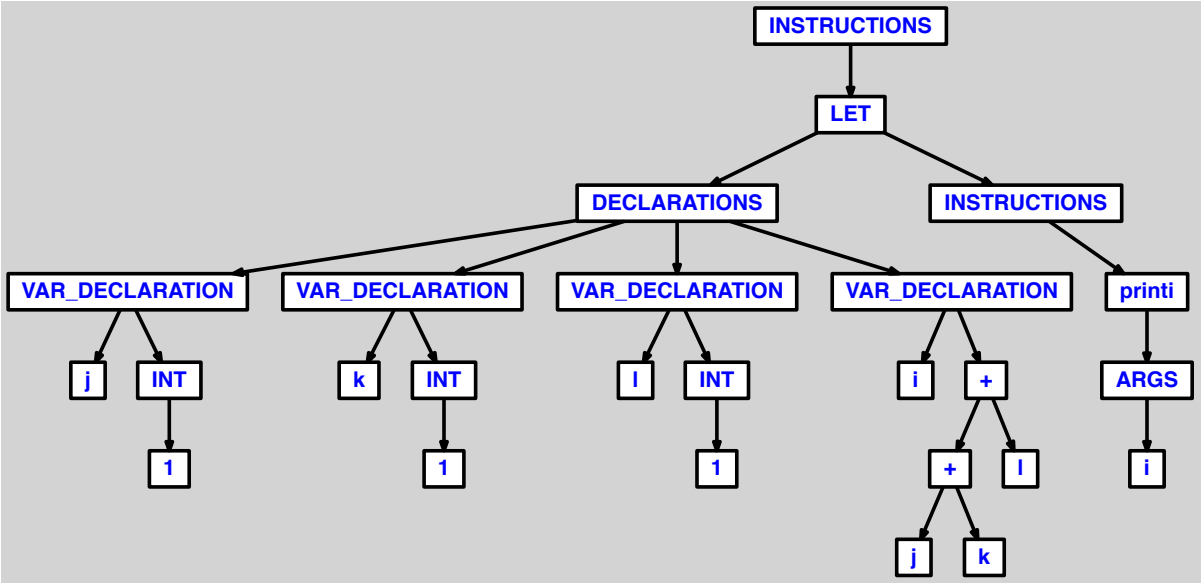
12.2.2 declarations d'entier avec reutilisation de 2 autres entiers

```
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
printi
j
INT
k
INT
i
+
ARGS
1
1
j
k
i

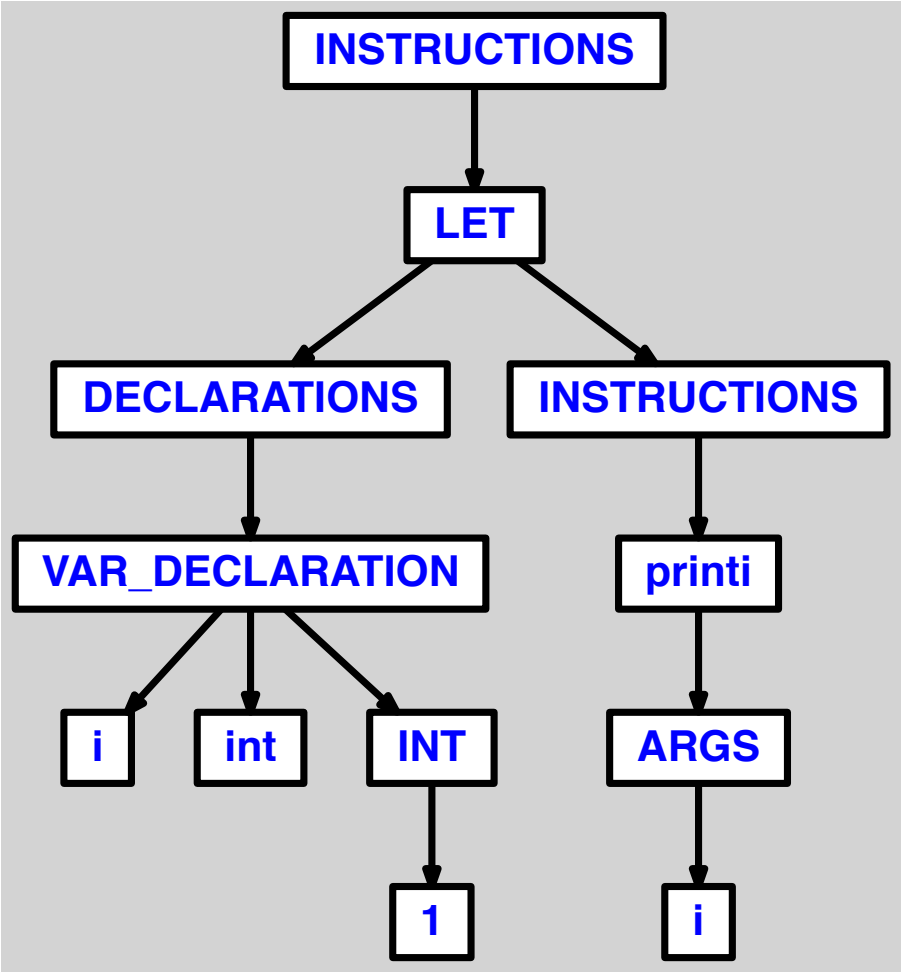
12.2.3 declarations d'entier avec reutilisation de 3 autres entiers

```
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
printi
j
INT
k
INT
l
INT
i
+
ARGS
1
1
1
+
l
i
j
k

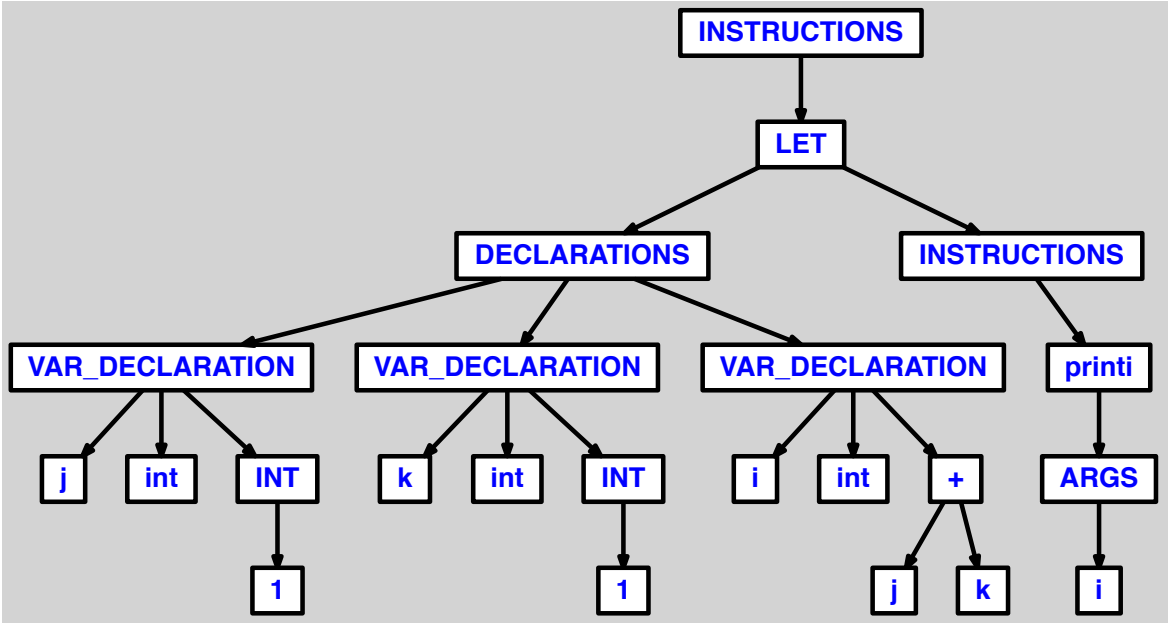
12.2.4 declaration simple d'entier avec type

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


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

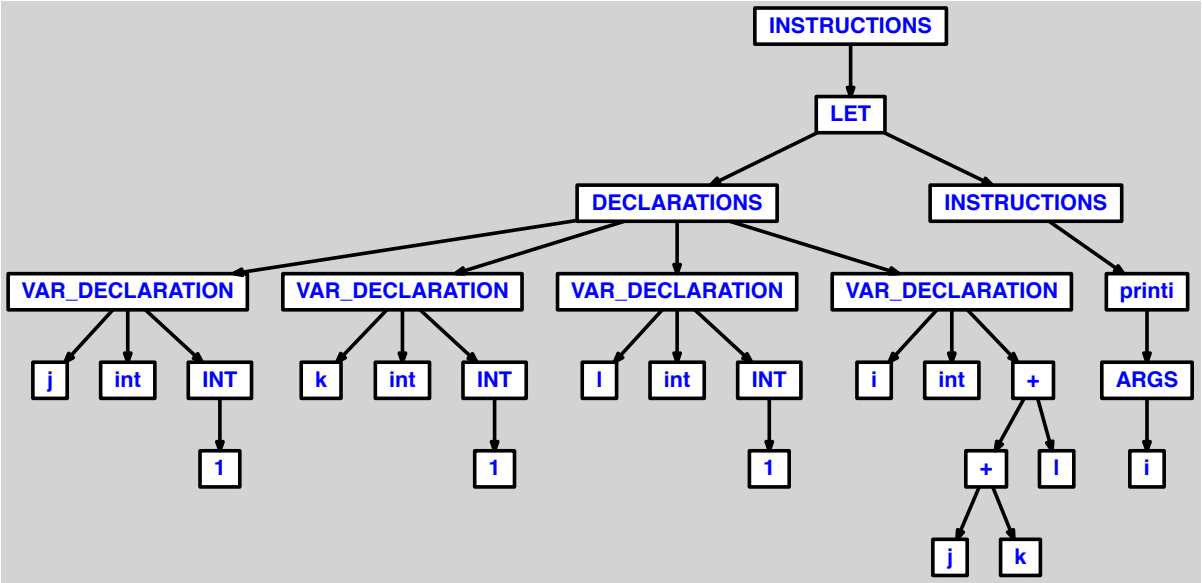
12.2.5 declarations d'entier avec types et reutilisation de 2 autres entiers

```
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
printi
j
int
INT
k
int
INT
i
int
+
ARGS
1
1
j
k
i

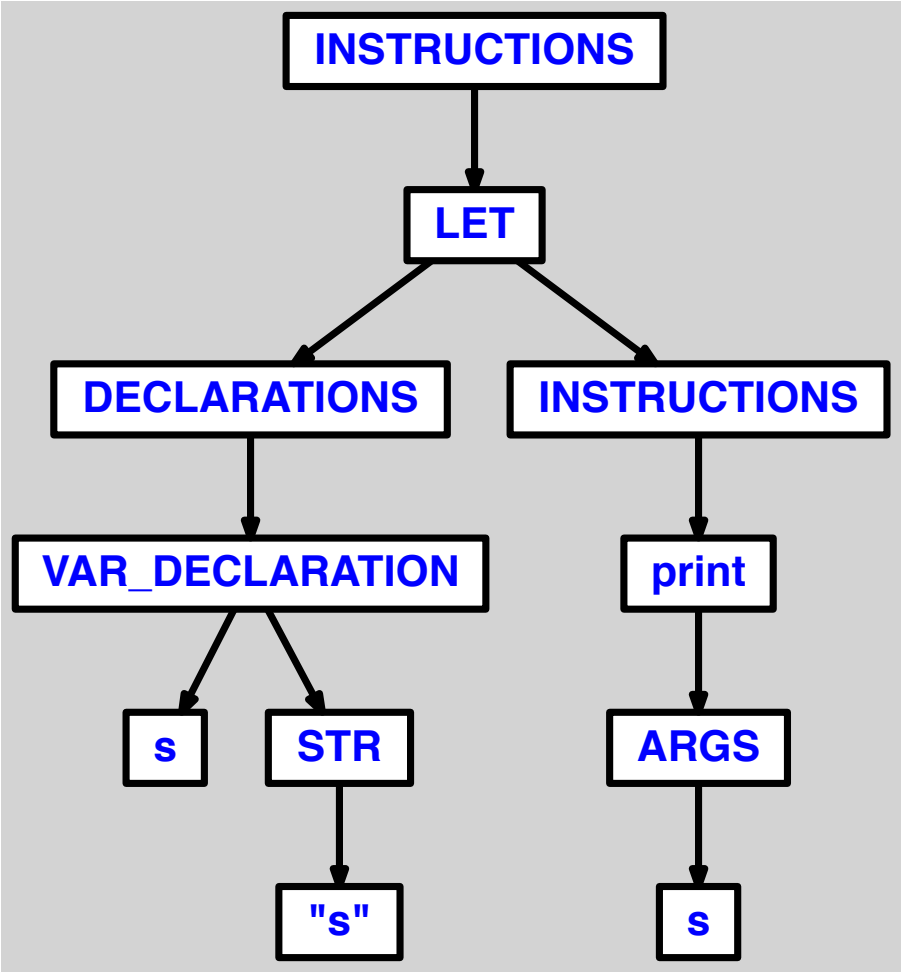
12.2.6 declarations d'entier avec types et reutilisation de 3 autres entiers

```
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
printi
j
int
INT
k
int
INT
l
int
INT
i
int
+
ARGS
1
1
1
+
l
i
j
k

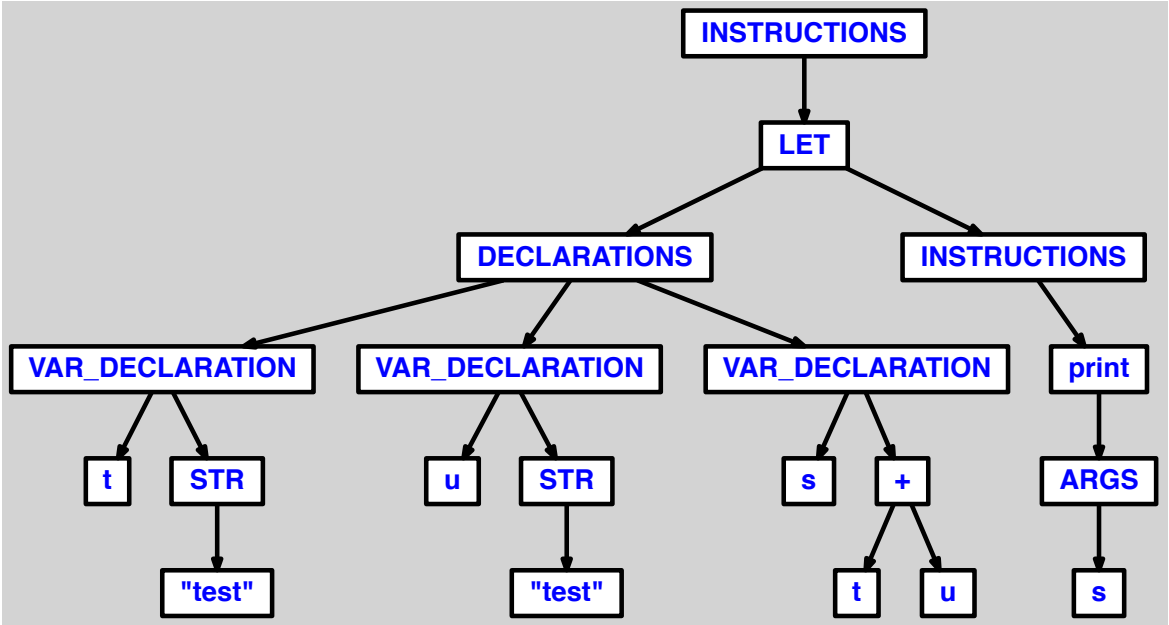
12.2.7 declaration de chaine simple

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


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

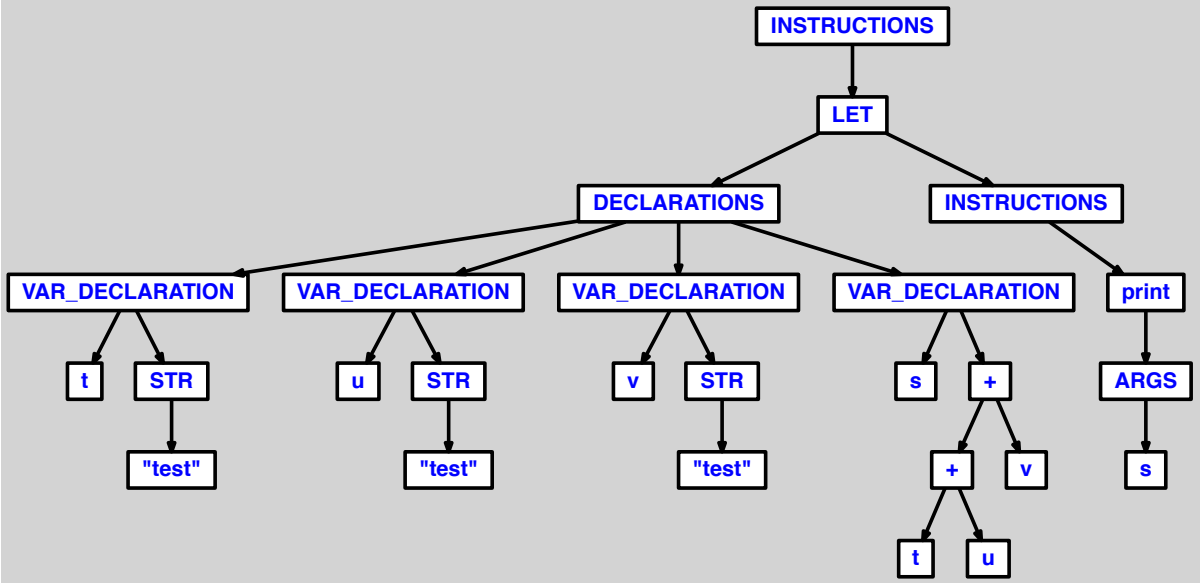
12.2.8 declarations de chaine avec reutilisation de 2 autres chaines

```
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
print
t
STR
u
STR
s
+
ARGS
"test"
"test"
t
u
s

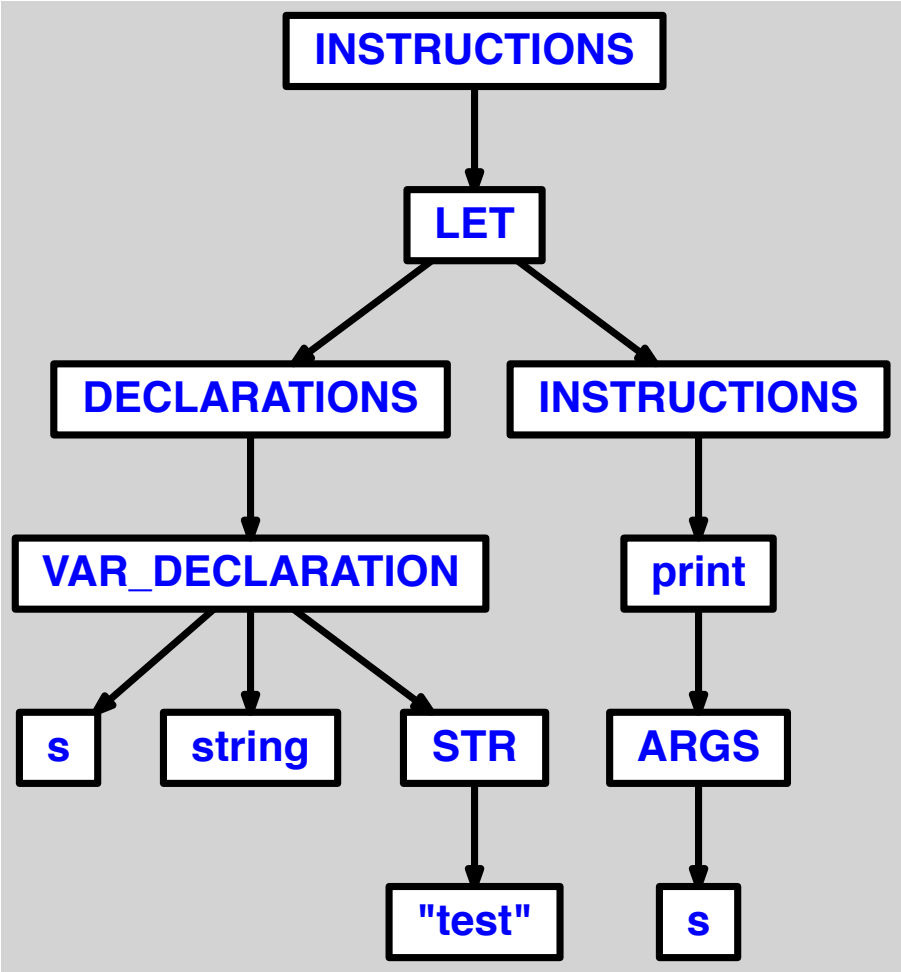
12.2.9 declarations de chaine avec reutilisation de 3 autres chaines

```
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
print
t
STR
u
STR
v
STR
s
+
ARGS
"test"
"test"
"test"
+
v
s
t
u

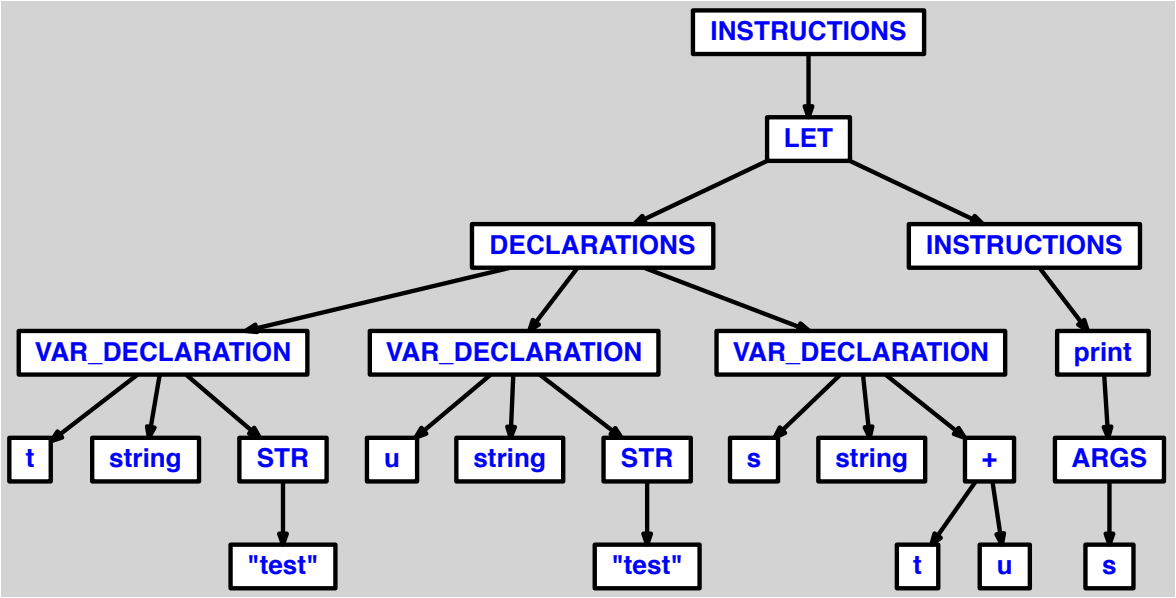
12.2.10 declaration de chaine avec type

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


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

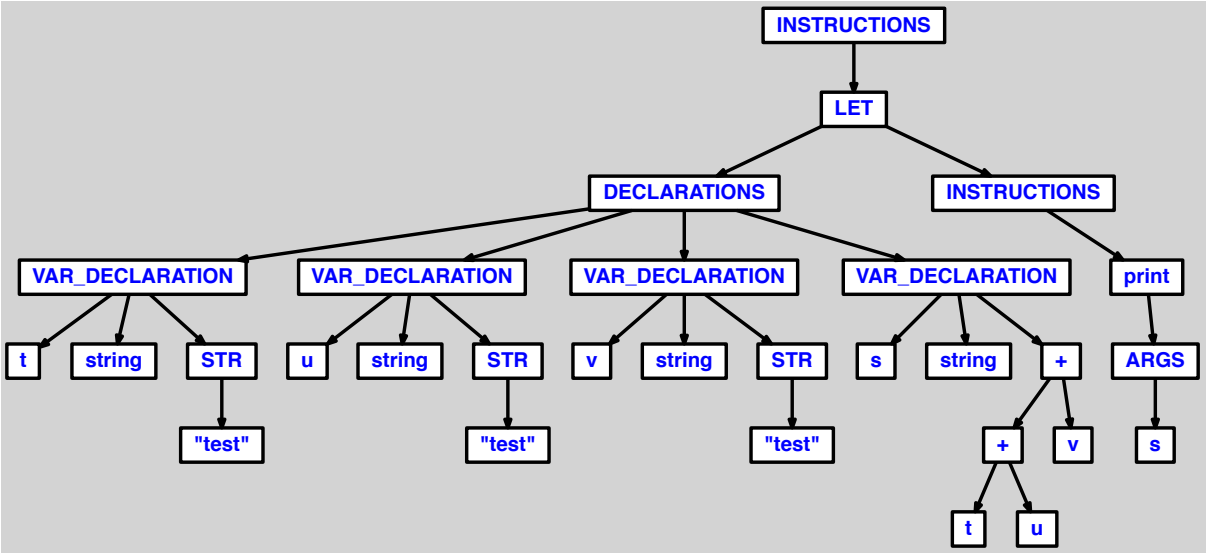
12.2.11 declarations de chaine avec types et reutilisation de 2 autres chaines

```
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
print
t
string
STR
u
string
STR
s
string
+
ARGS
"test"
"test"
t
u
s

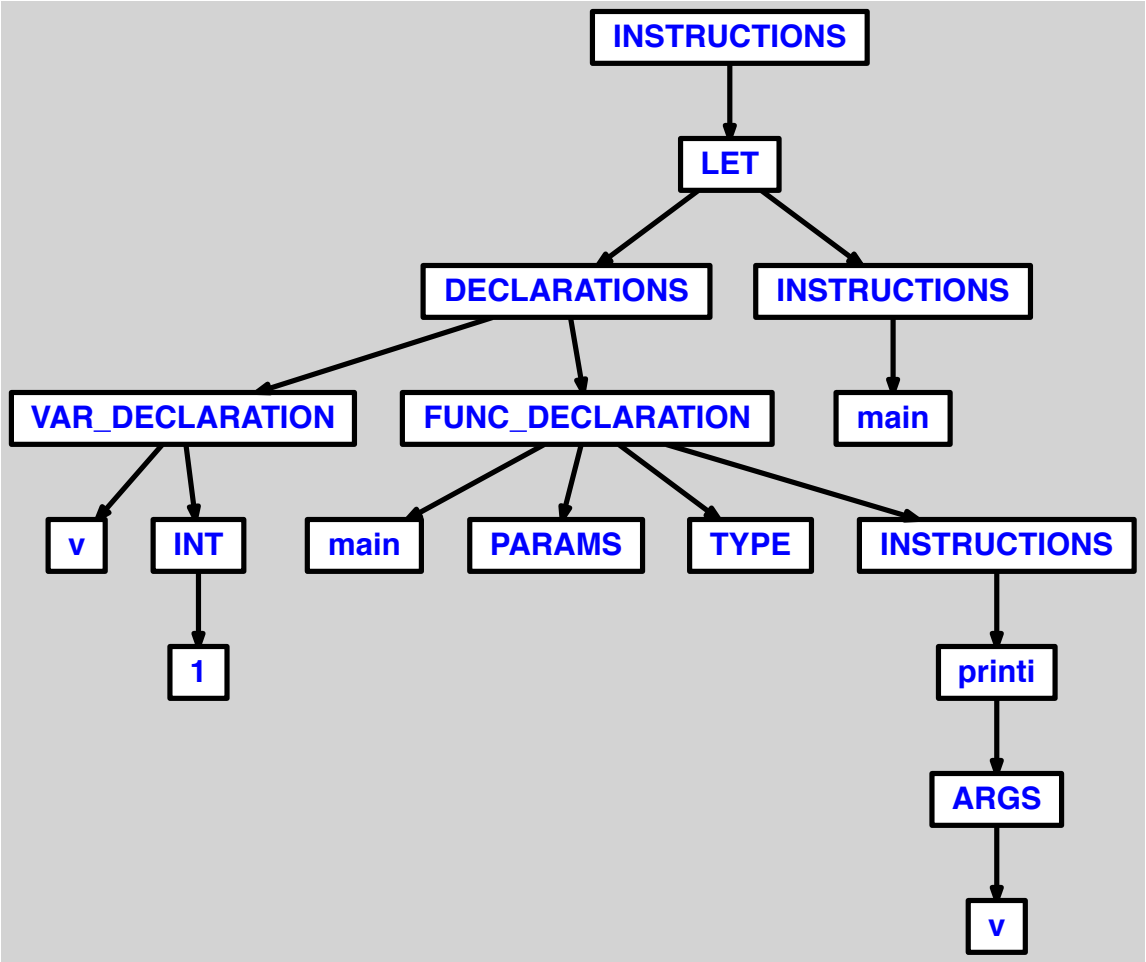
12.2.12 declarations de chaine avec types et reutilisation de 3 autres chaines

```
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
print
t
string
STR
u
string
STR
v
string
STR
s
string
+
ARGS
"test"
"test"
"test"
+
v
s
t
u

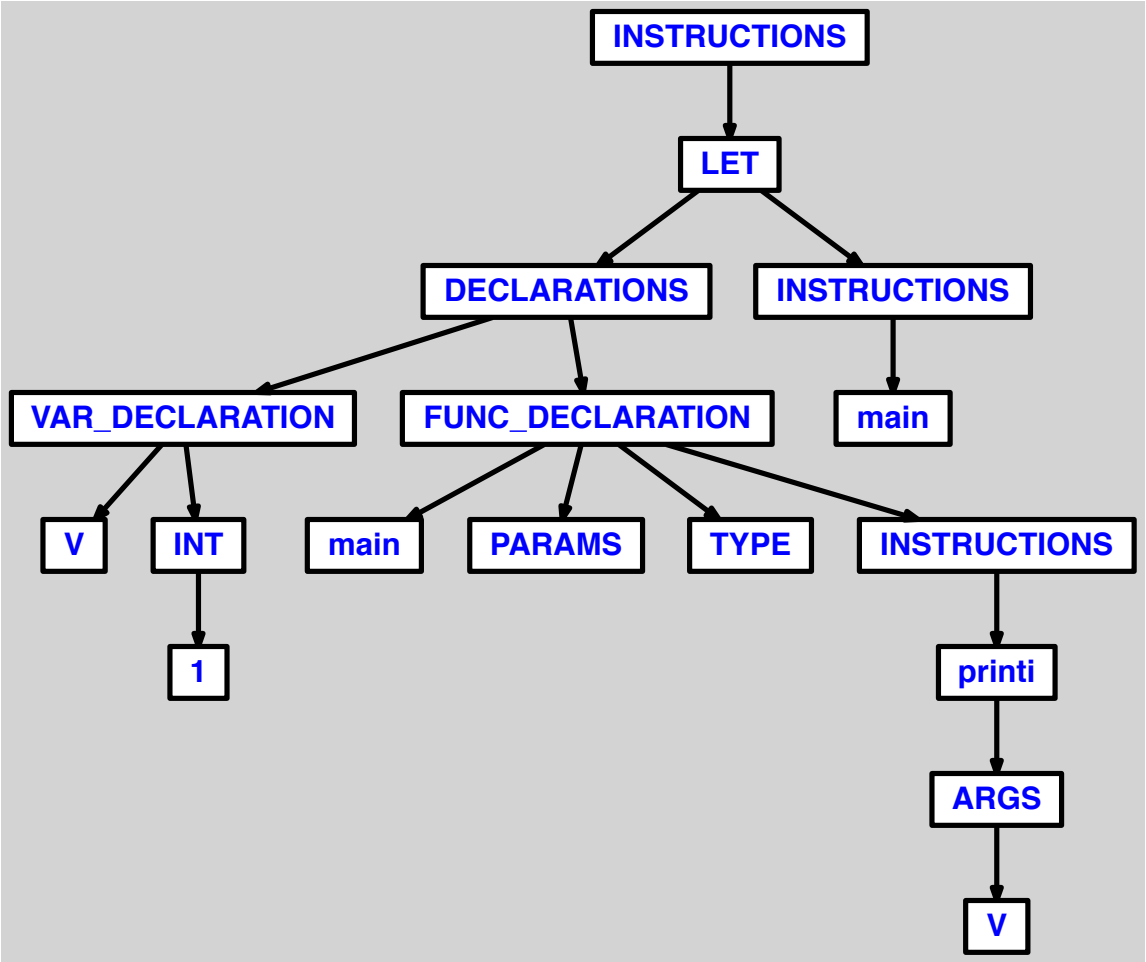
12.2.13 lettre minuscule comme nom de variable

```
let
  var v := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
main
v
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
printi
ARGS
v

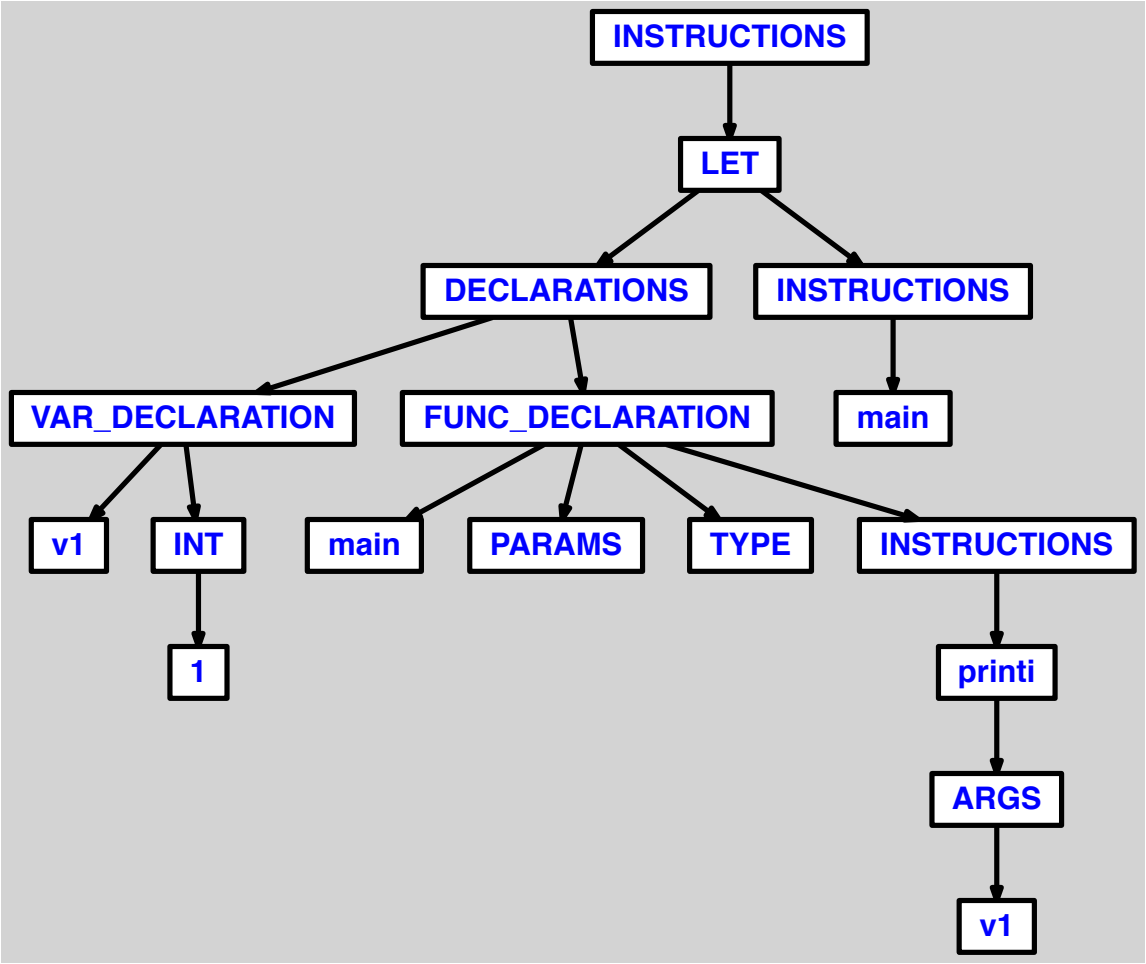
12.2.14 lettre majuscule comme nom de variable

```
let
  var V := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
main
V
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
printi
ARGS
V

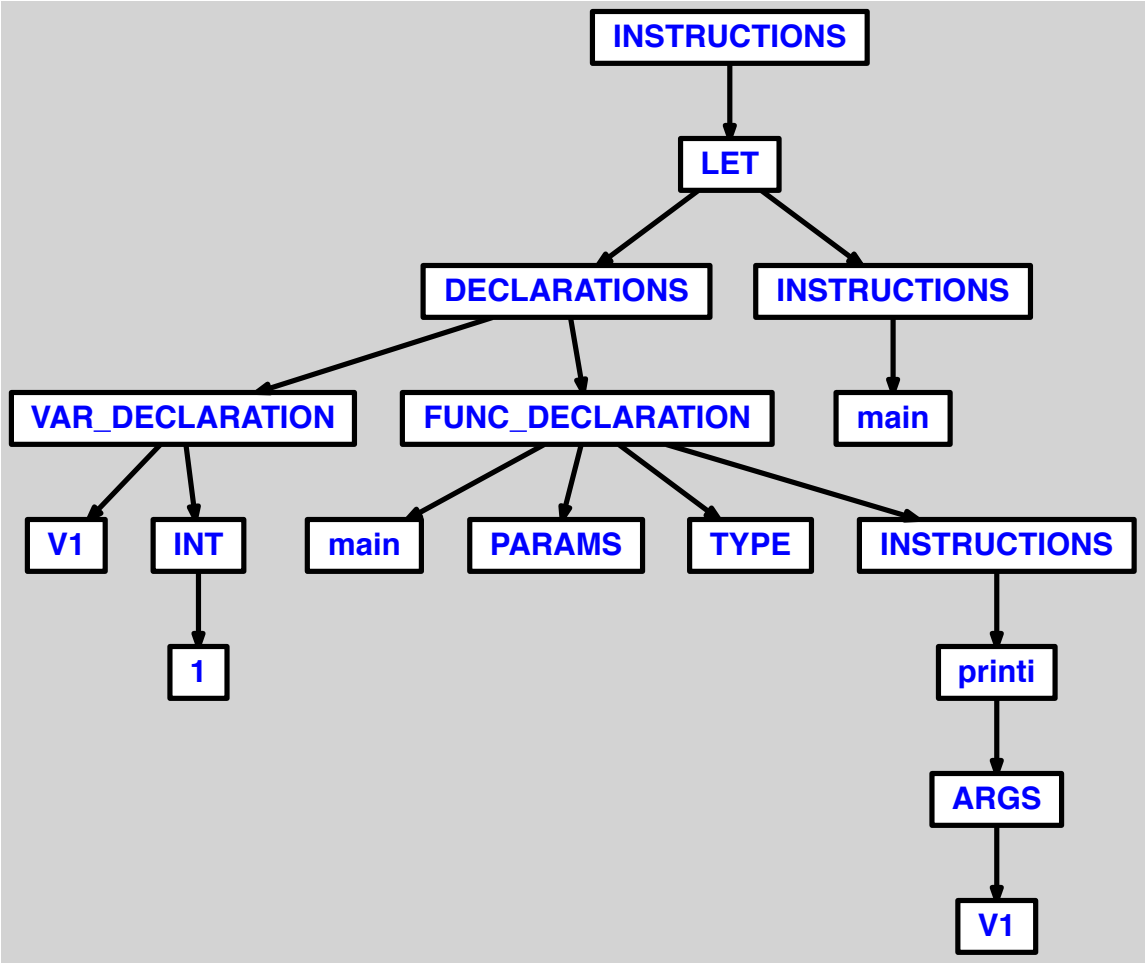
12.2.15 lettre minuscule et chiffre dans nom de variable

```
let
  var v1 := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
main
v1
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
printi
ARGS
v1

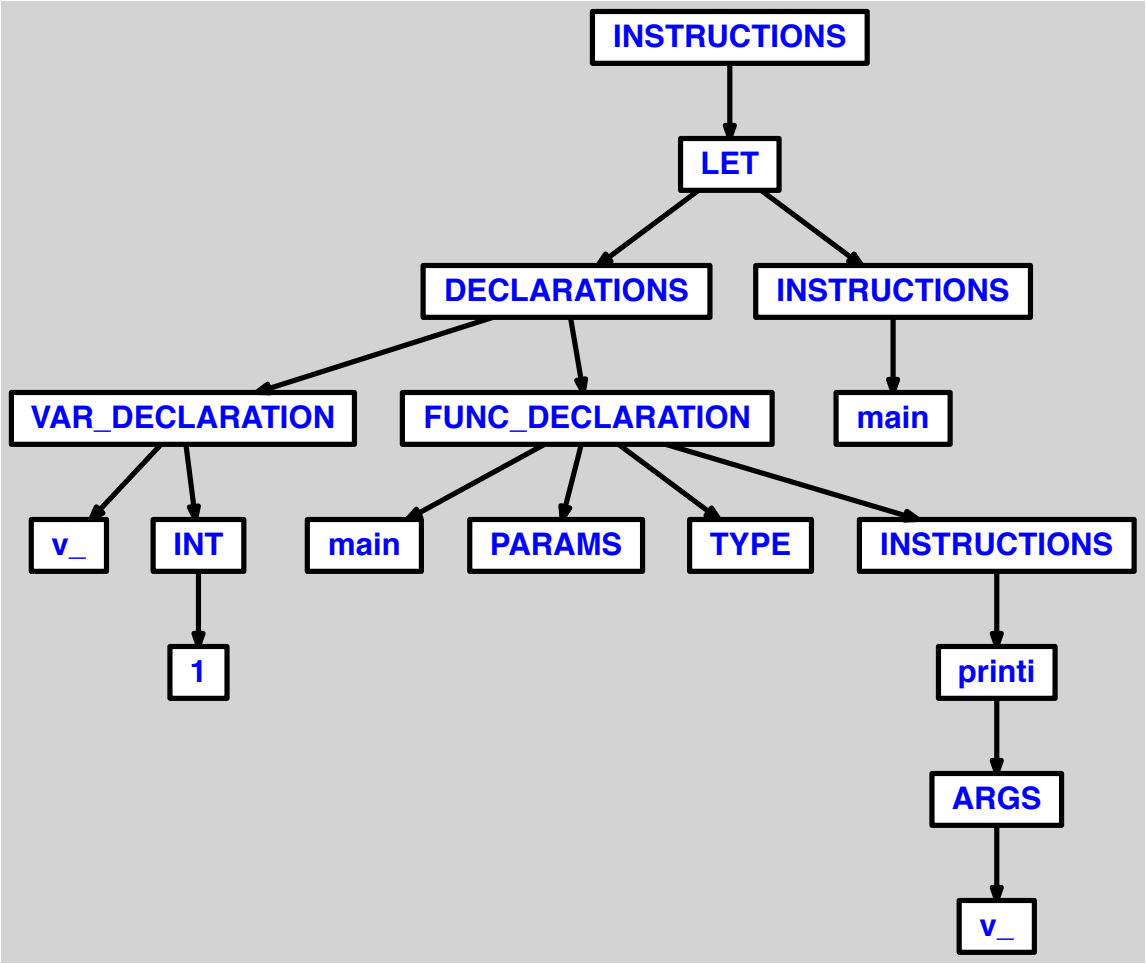
12.2.16 lettre majuscule et chiffre dans nom de variable

```
let
  var V1 := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
main
V1
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
printi
ARGS
V1

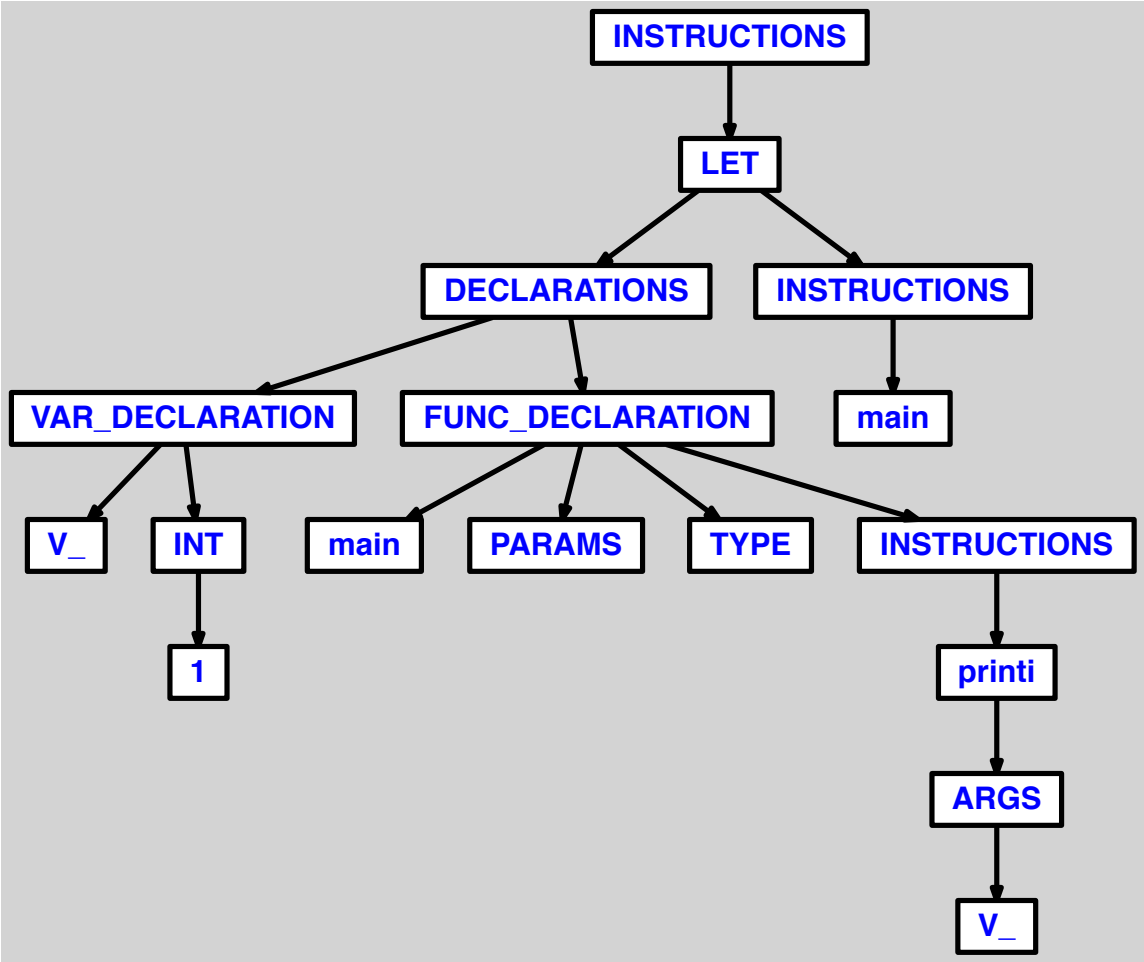
12.2.17 lettre minuscule et <_> dans nom de variable

```
let
  var v_ := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
main
v_
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
printi
ARGS
v_

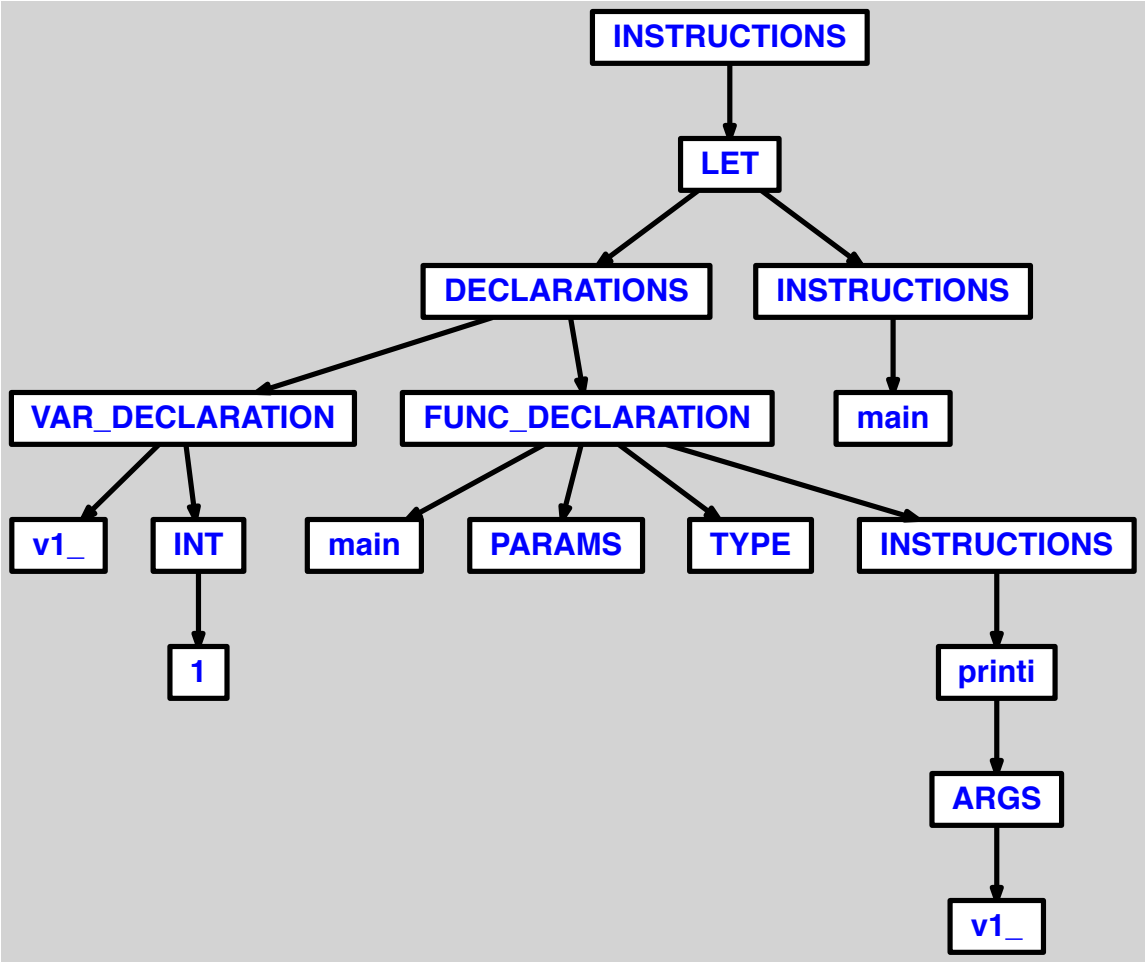
12.2.18 lettre majuscule et <_> dans nom de variable

```
let
  var V_ := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
main
V_
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
printi
ARGS
V_

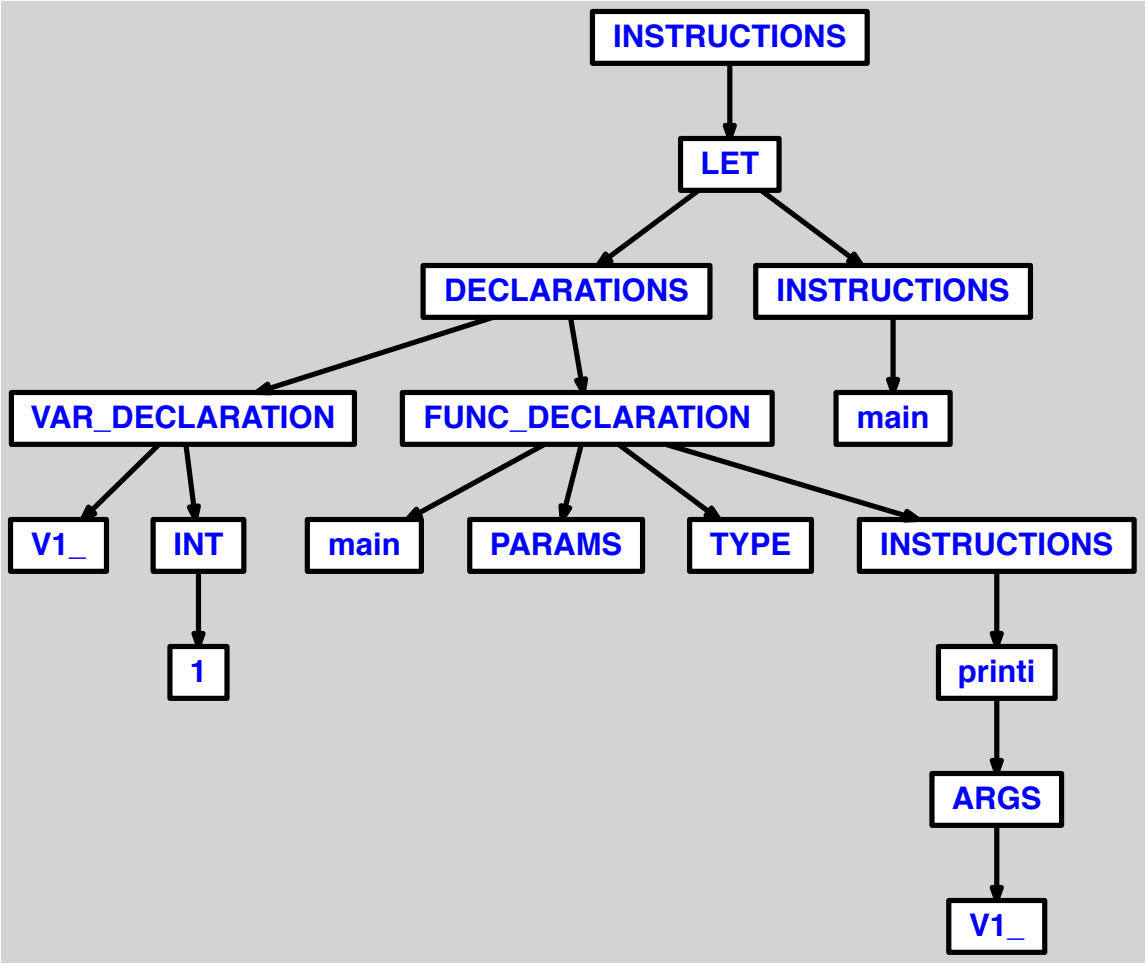
12.2.19 lettre minuscule, chiffre et <_> dans nom de variable

```
let
  var v1_ := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
main
v1_
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
printi
ARGS
v1_

12.2.20 lettre majuscule, chiffre et <_> dans nom de variable

```
let
  var V1_ := 1

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

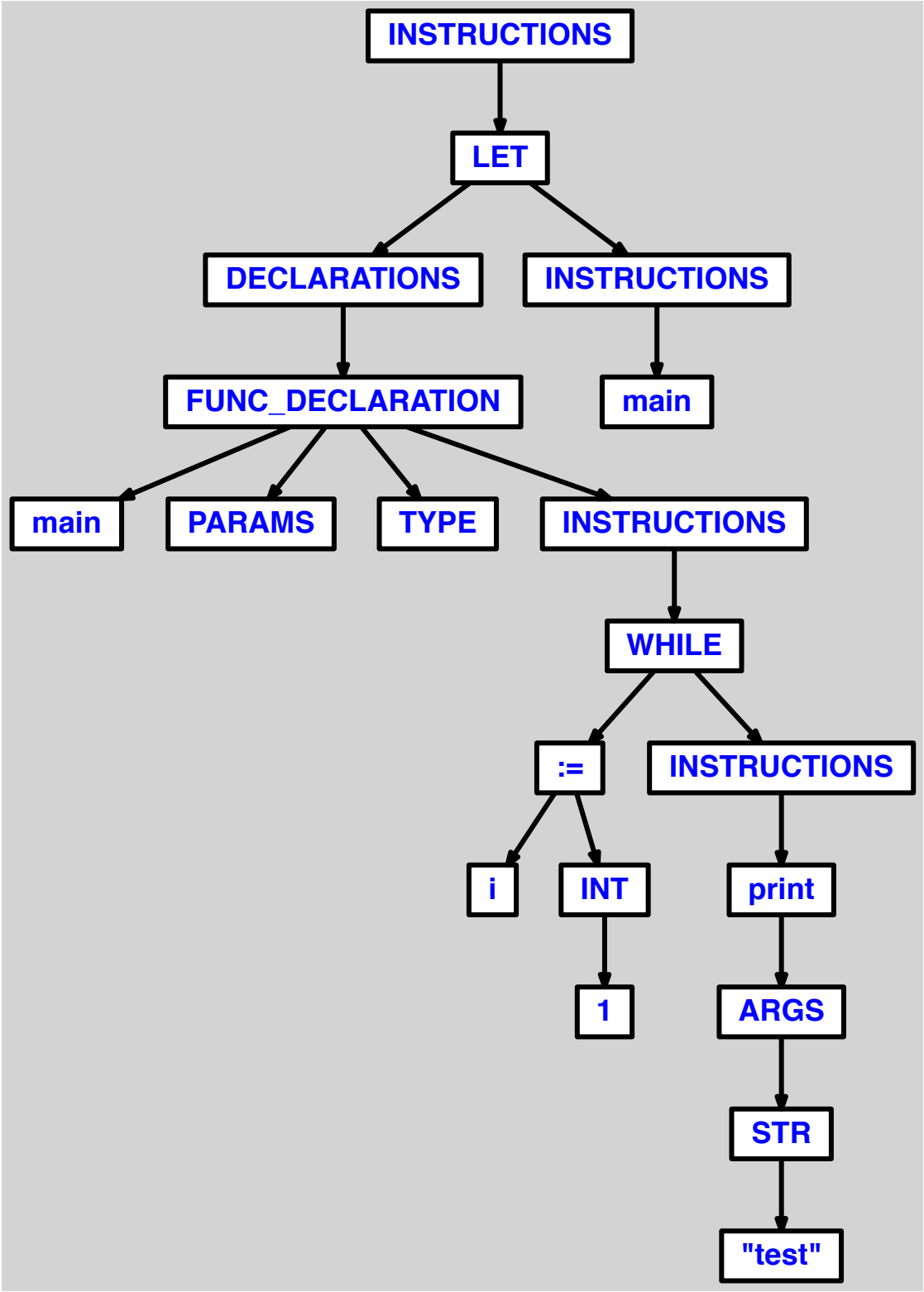

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
main
V1_
INT
main
PARAMS
TYPE
INSTRUCTIONS
1
printi
ARGS
V1_

13 while

13.1 KO

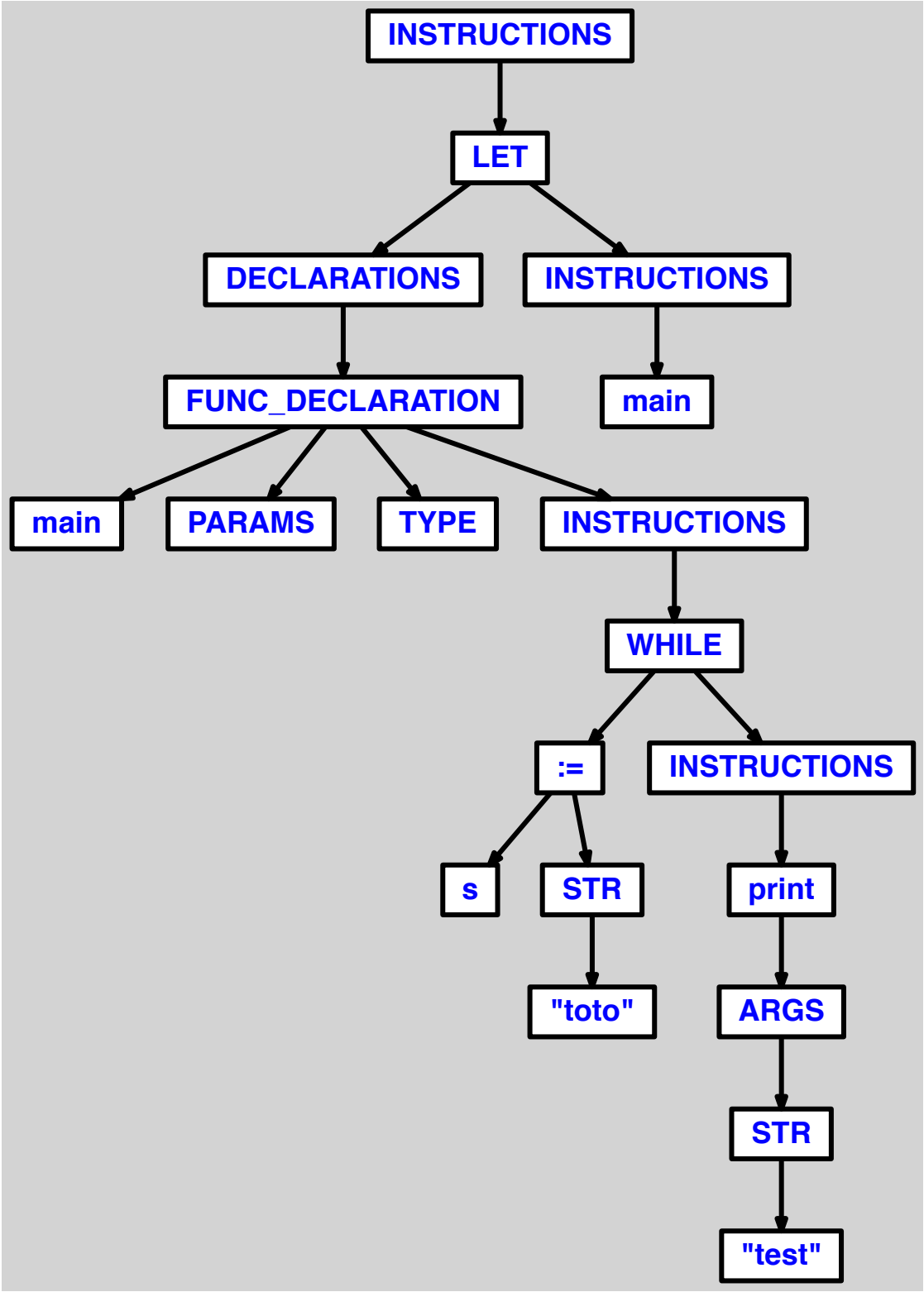
13.1.1 <while> avec affectation d'entier

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
WHILE
:=
INSTRUCTIONS
i
INT
print
1
ARGS
STR
"test"

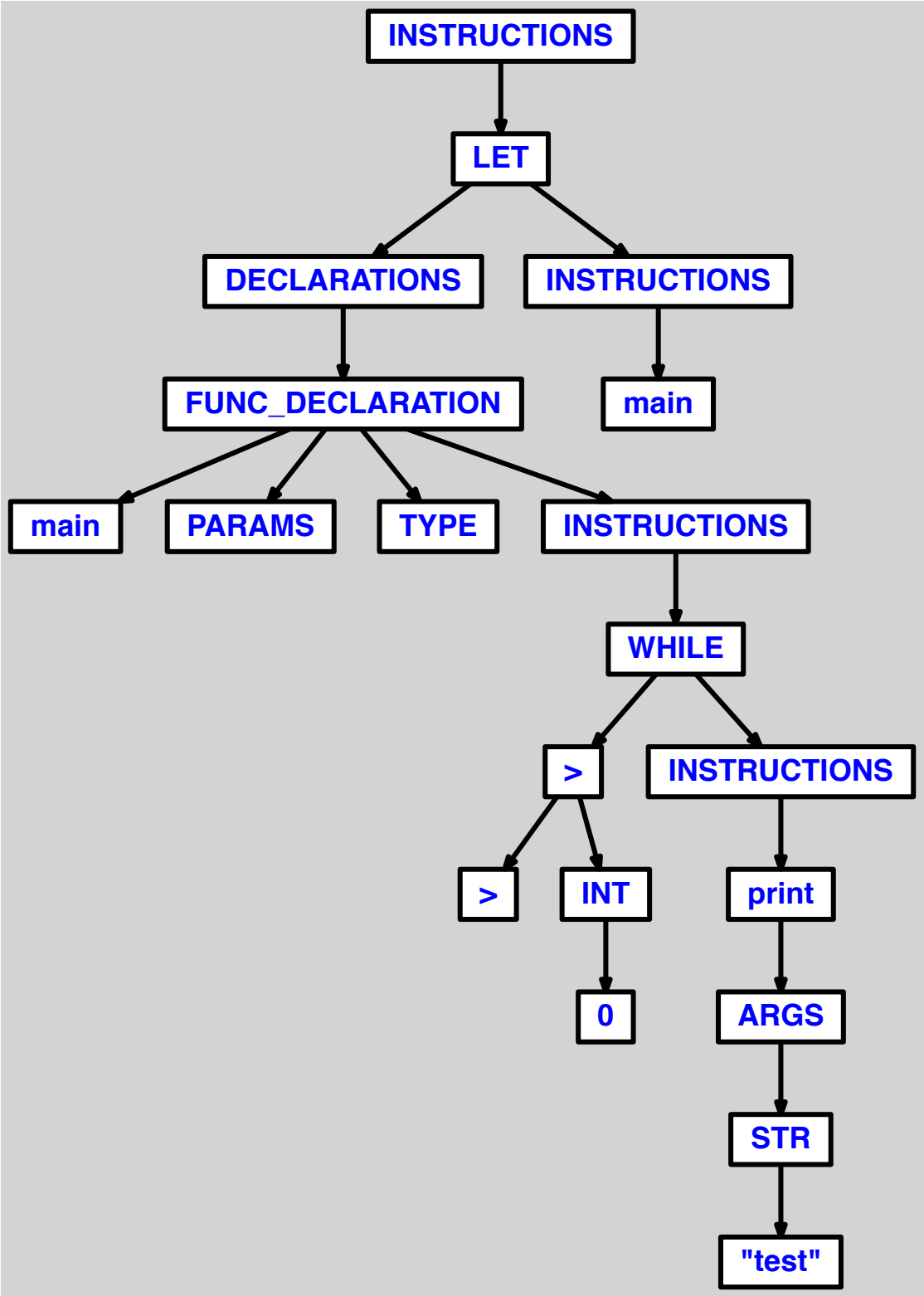
13.1.2 <while> avec affectation de chaine

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
WHILE
:=
INSTRUCTIONS
s
STR
print
"toto"
ARGS
STR
"test"

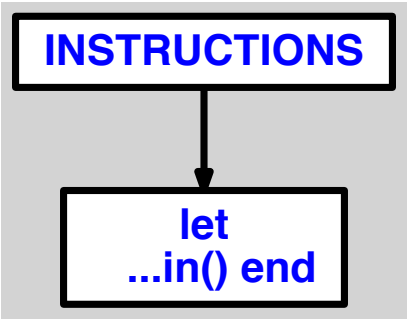
13.1.3 <while> avec oubli du compteur

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
WHILE
>
INSTRUCTIONS
>
INT
print
0
ARGS
STR
"test"

13.1.4 <while> avec oubli du <while>

```
let
  var i := 0

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


INSTRUCTIONS
let
...in() end

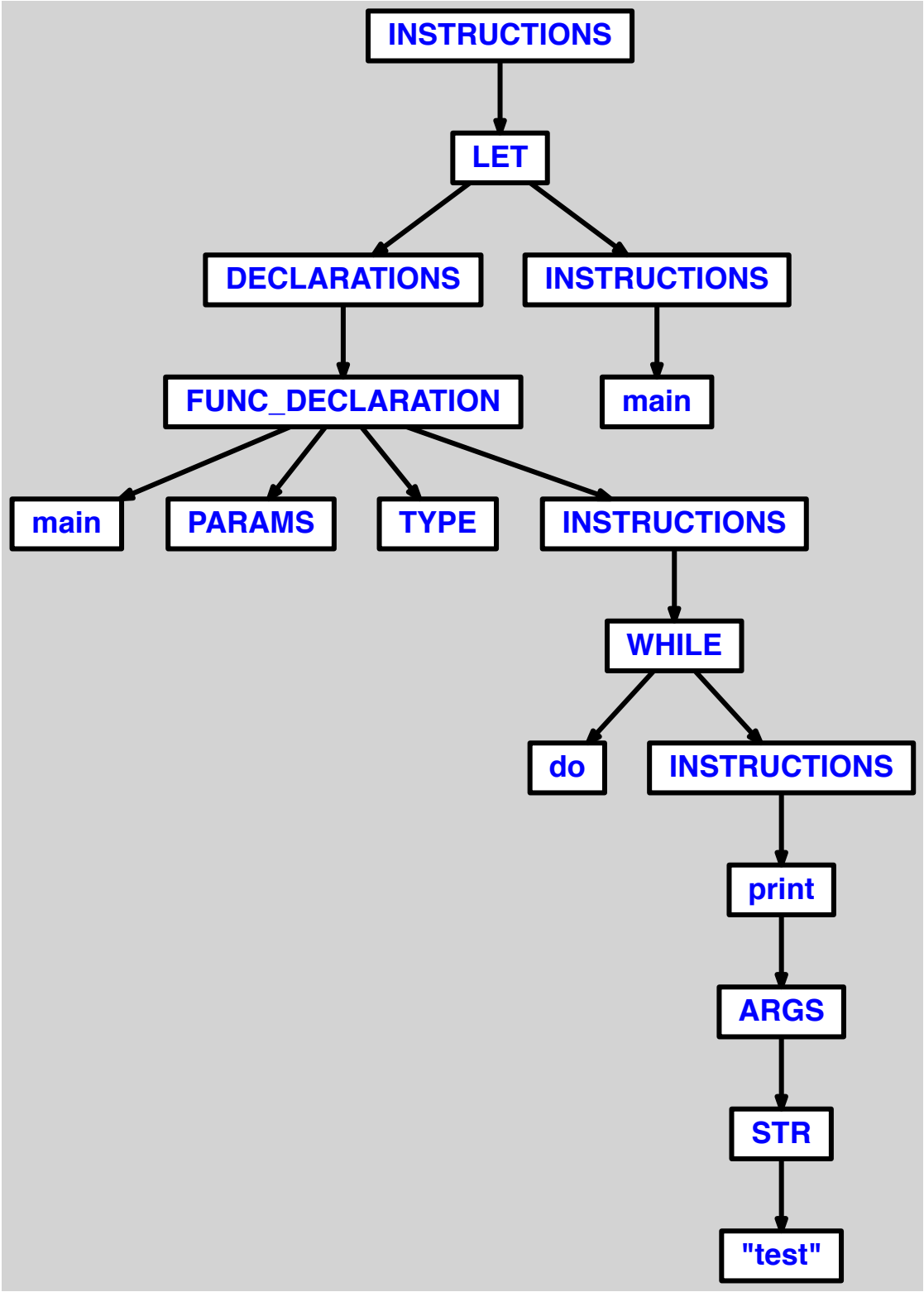
13.1.5 <while> avec oubli du <do>

```
let
  var i := 0

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

Pas d'AST, problème de syntaxe.

13.1.6 <while> avec oubli de la condition

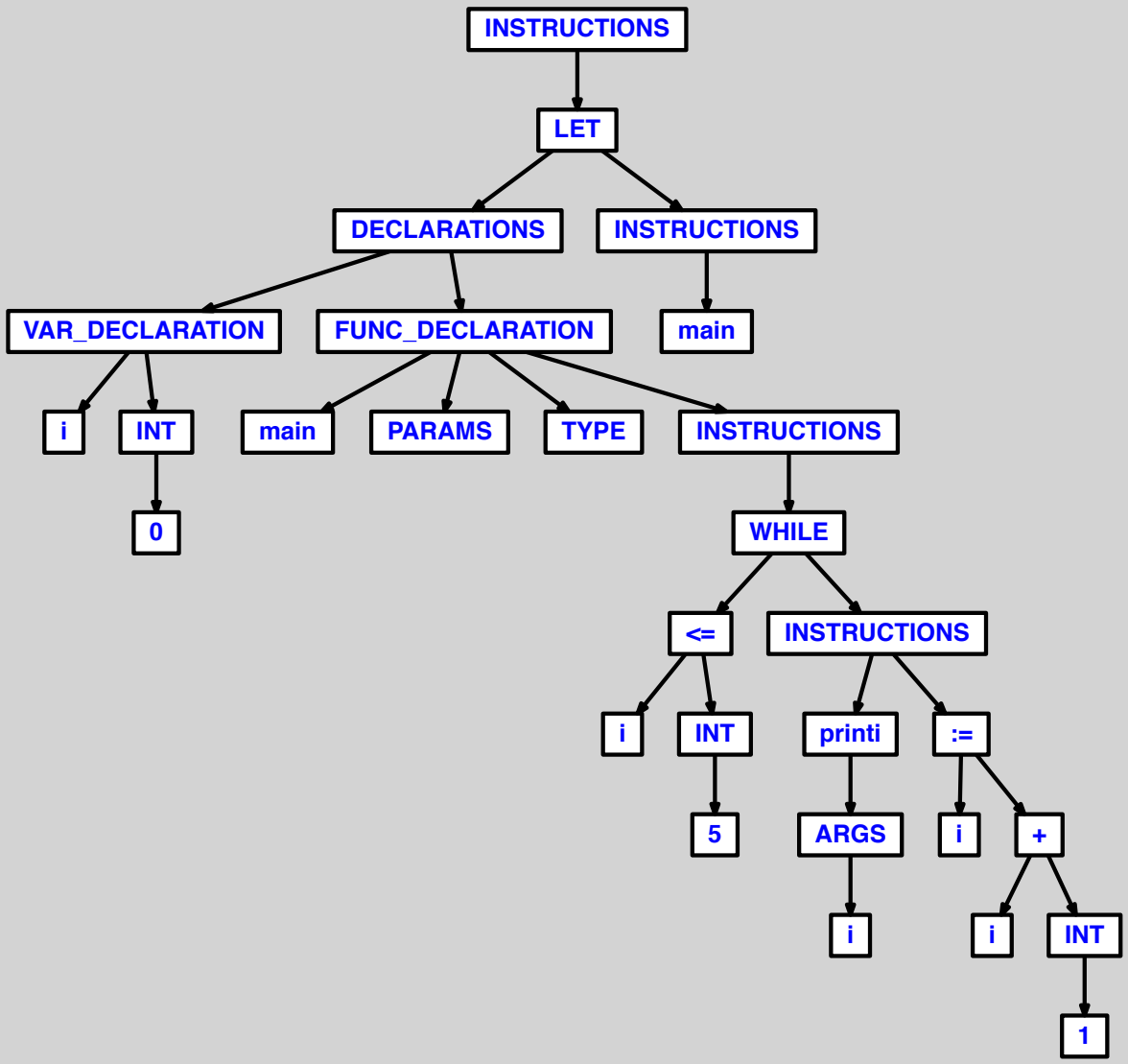
```
let
  function main() =
    while do
      print("test")
in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
main
main
PARAMS
TYPE
INSTRUCTIONS
WHILE
do
INSTRUCTIONS
print
ARGS
STR
"test"

13.2 OK

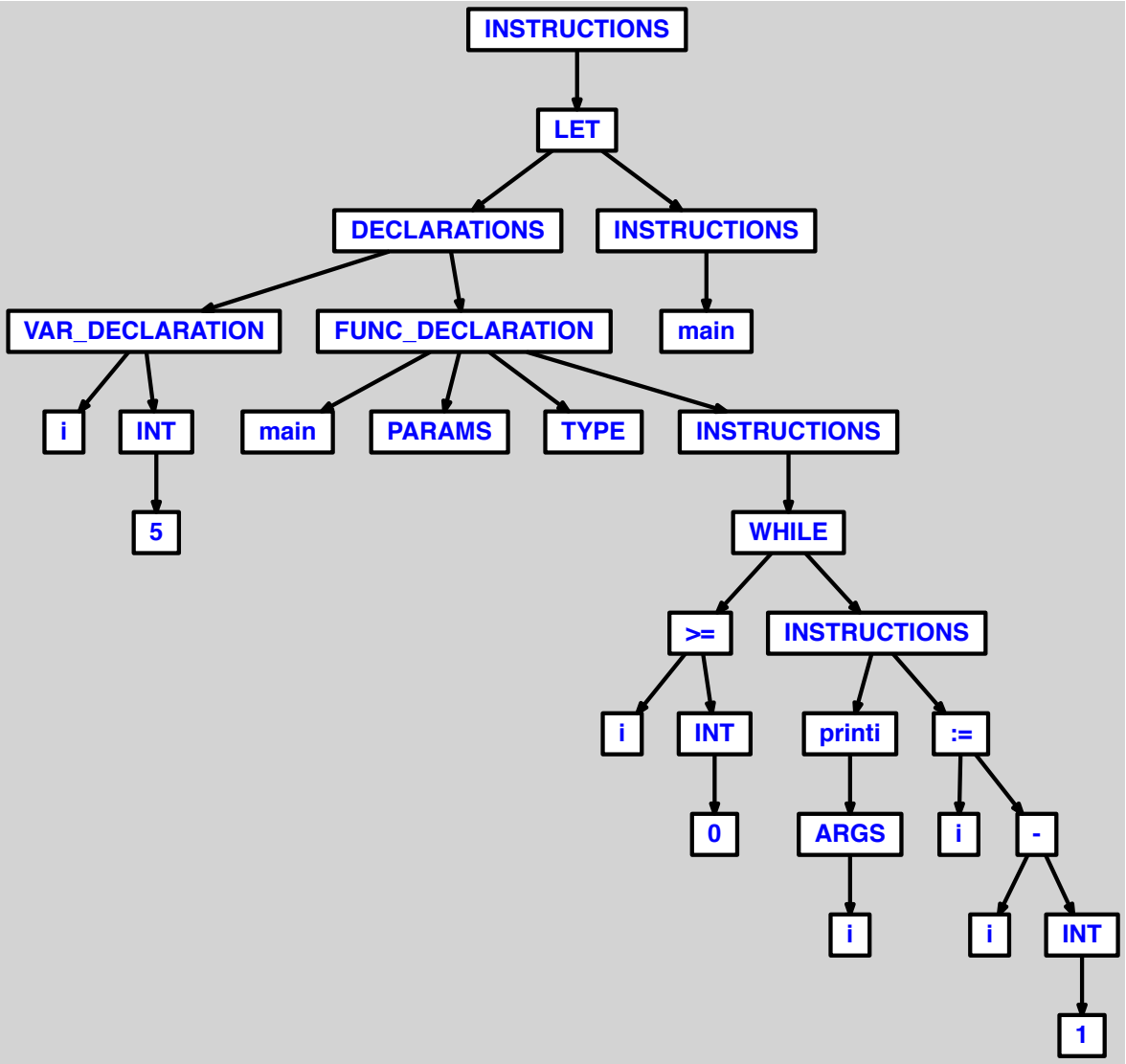
13.2.1 <while> avec iteration croissante

```
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
main
i
INT
main
PARAMS
TYPE
INSTRUCTIONS
0
WHILE
<=
INSTRUCTIONS
i
INT
printi
:=
5
ARGS
i
+
i
i
INT
1

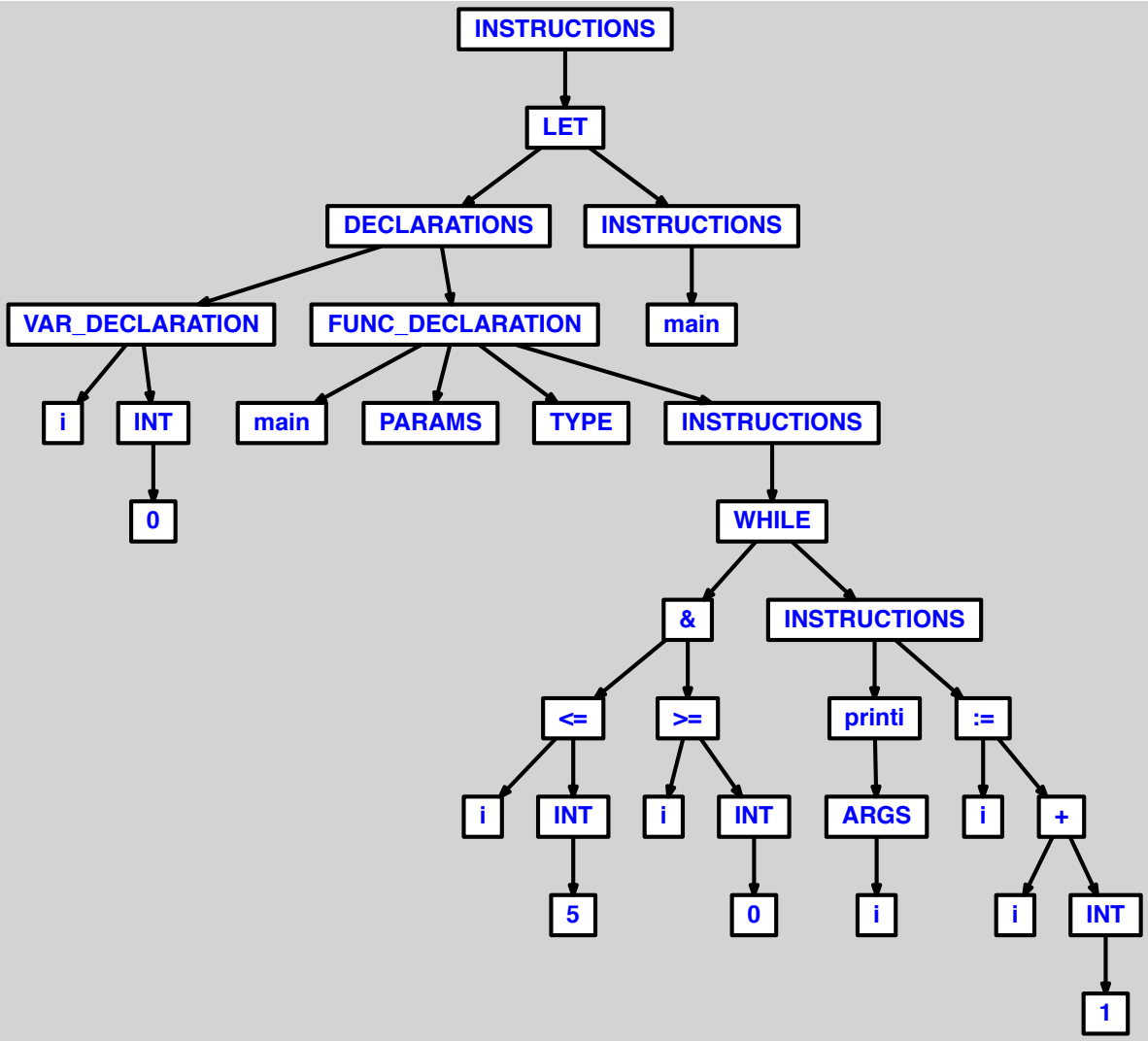
13.2.2 <while> avec iteration decroissante

```
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
main
i
INT
main
PARAMS
TYPE
INSTRUCTIONS
5
WHILE
>=
INSTRUCTIONS
i
INT
printi
:=
0
ARGS
i
-
i
i
INT
1

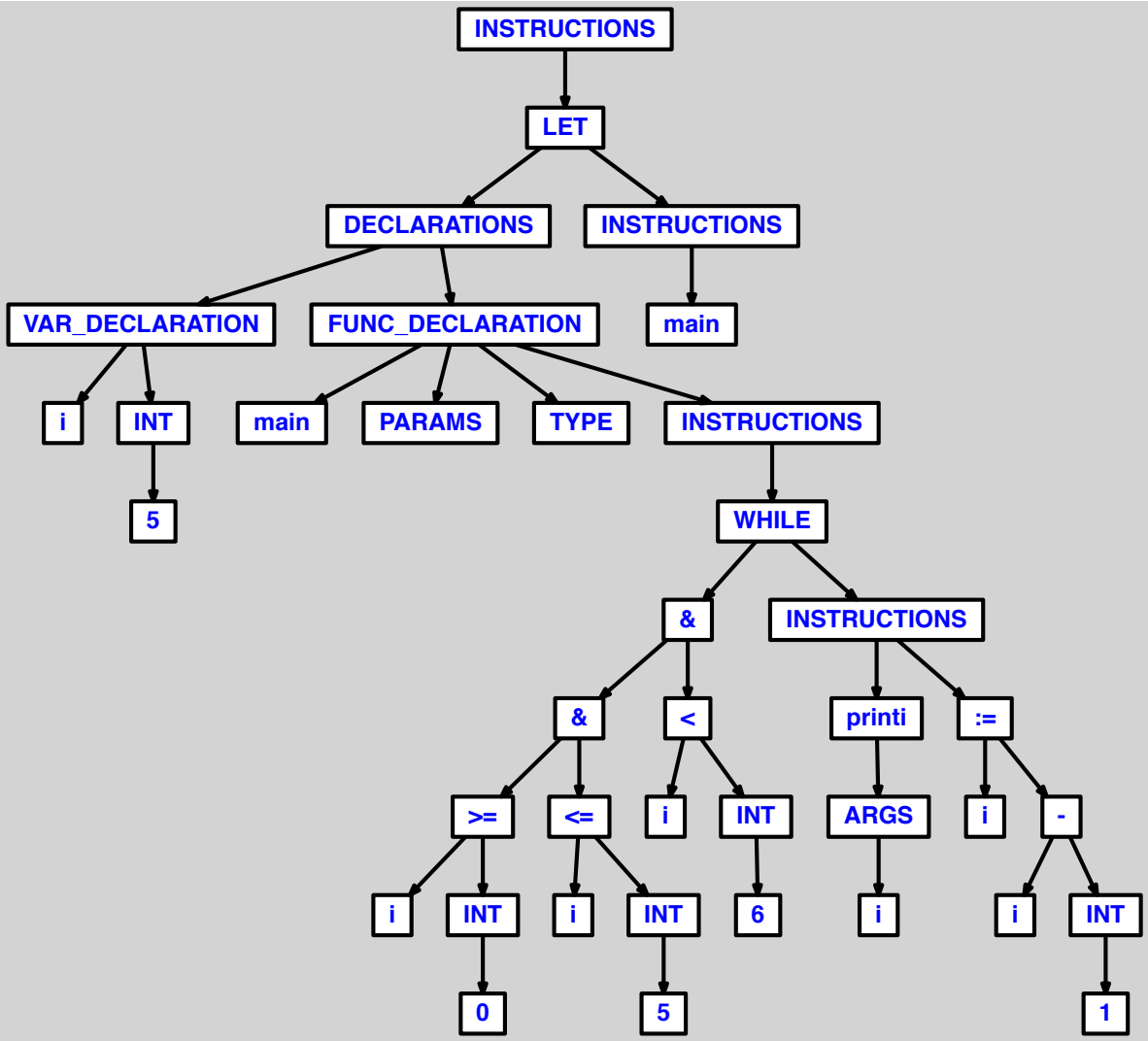
13.2.3 <while> avec double-condition

```
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
main
i
INT
main
PARAMS
TYPE
INSTRUCTIONS
0
WHILE
&
INSTRUCTIONS
<=
>=
printi
:=
i
INT
i
INT
ARGS
i
+
5
0
i
i
INT
1

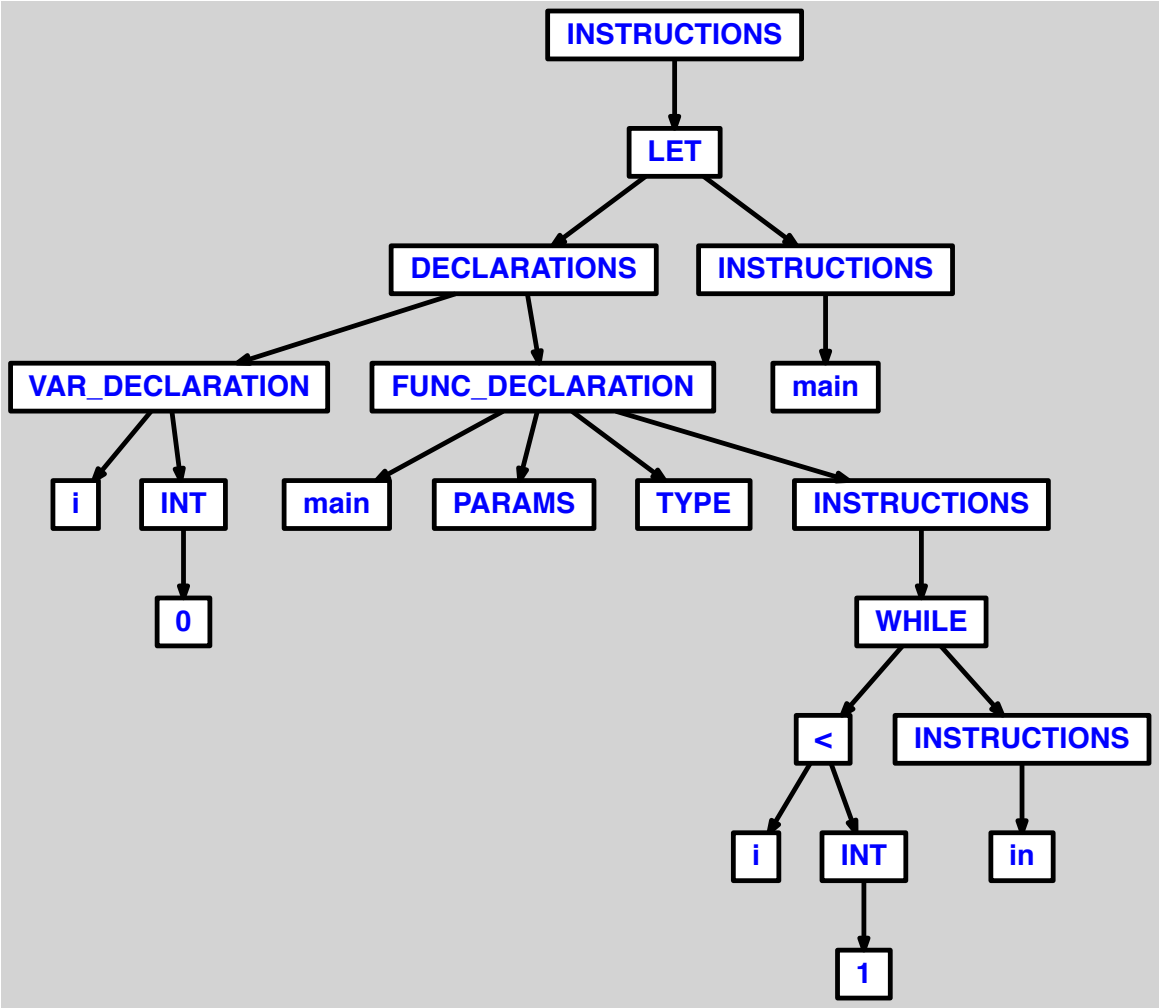
13.2.4 <while> avec triple-condition

```
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
main
i
INT
main
PARAMS
TYPE
INSTRUCTIONS
5
WHILE
&
INSTRUCTIONS
&
<
printi
:=
>=
<=
i
INT
ARGS
i
-
i
INT
i
INT
6
i
i
INT
0
5
1

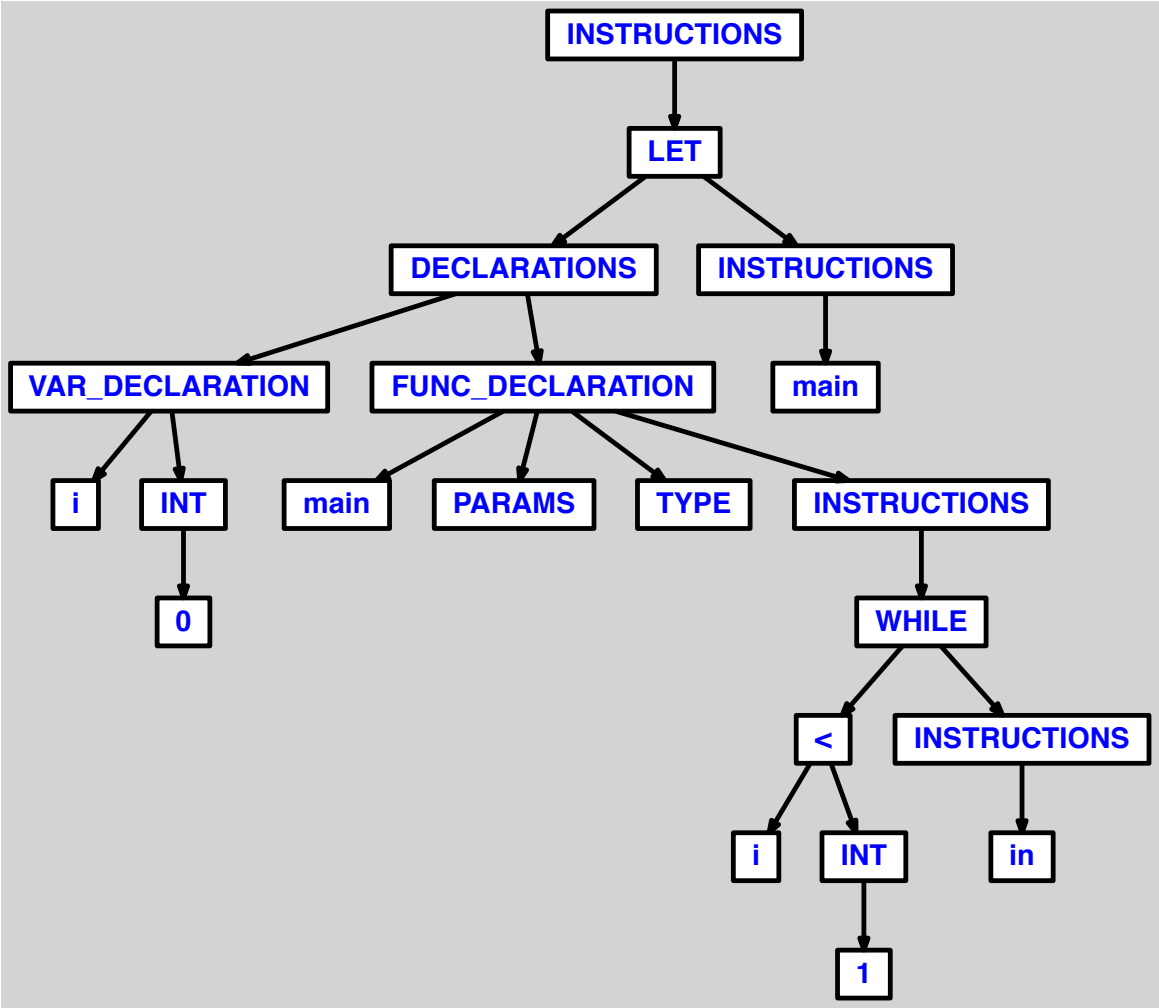
13.2.5 <while> sans instruction

```
let
  var i := 0

  function main() =
    while i < 1 do
in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
main
i
INT
main
PARAMS
TYPE
INSTRUCTIONS
0
WHILE
<
INSTRUCTIONS
i
INT
in
1

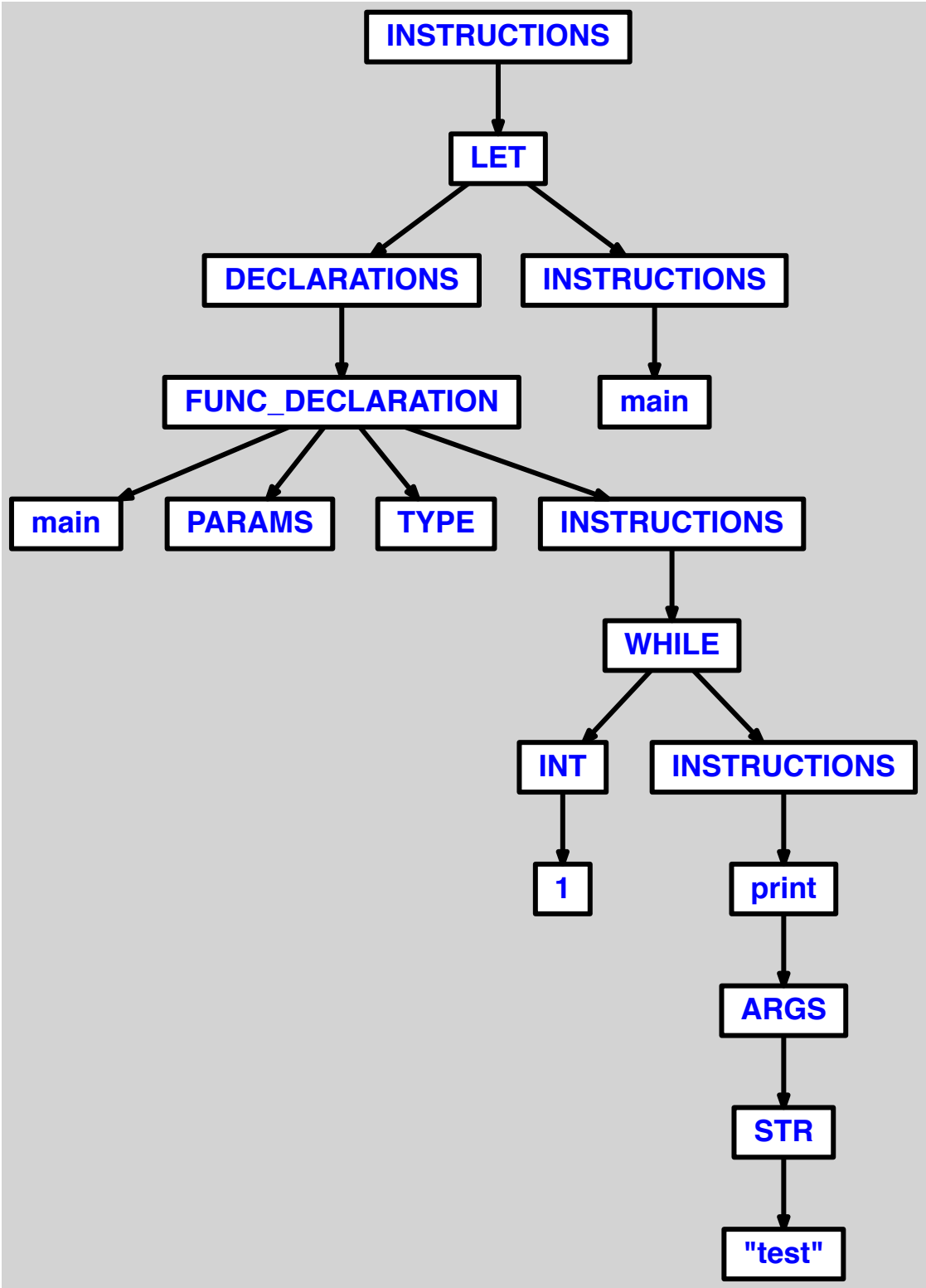
13.2.6 <while> avec ligne vide

```
let
  var i := 0

  function main() =
    while i < 1 do

in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
main
i
INT
main
PARAMS
TYPE
INSTRUCTIONS
0
WHILE
<
INSTRUCTIONS
i
INT
in
1

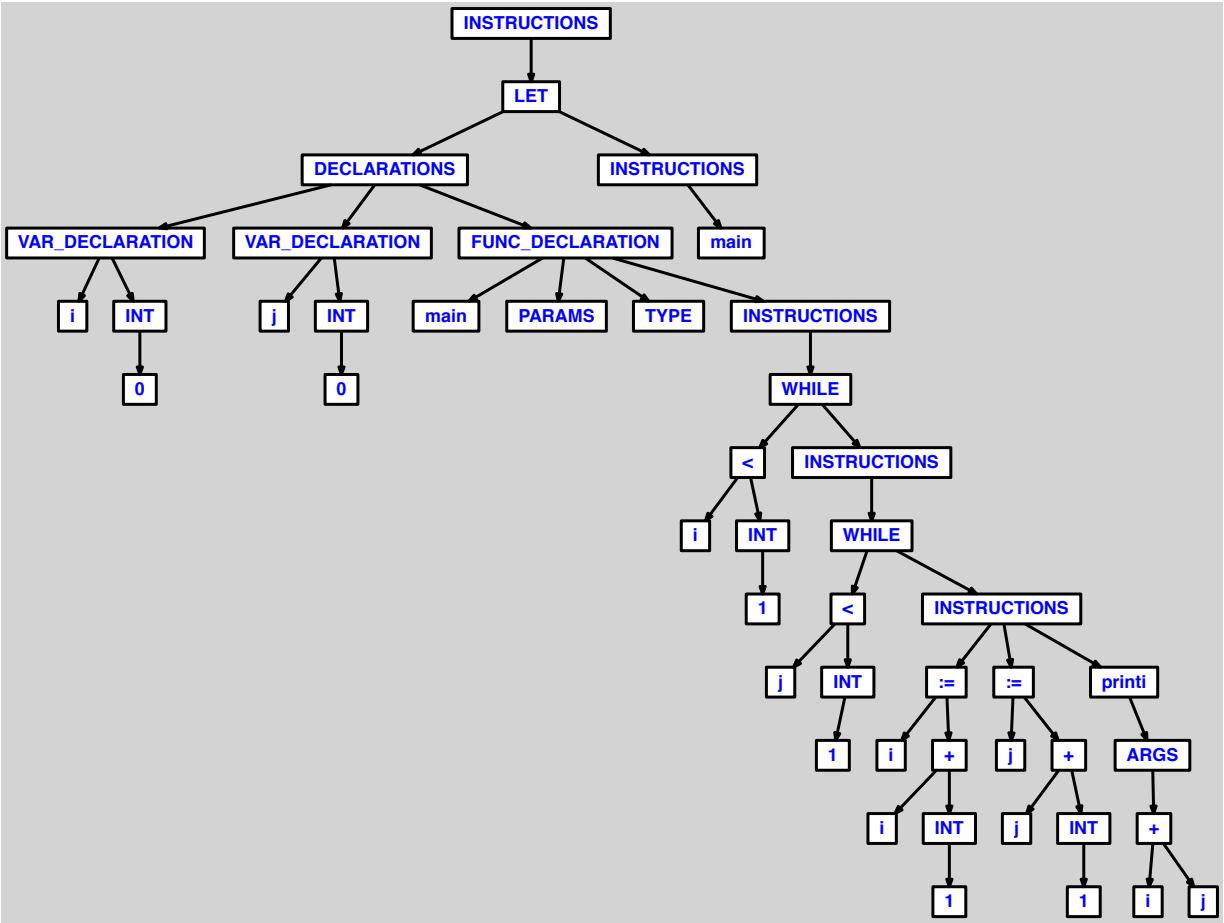
13.2.7 <while> avec condition entiere

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


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

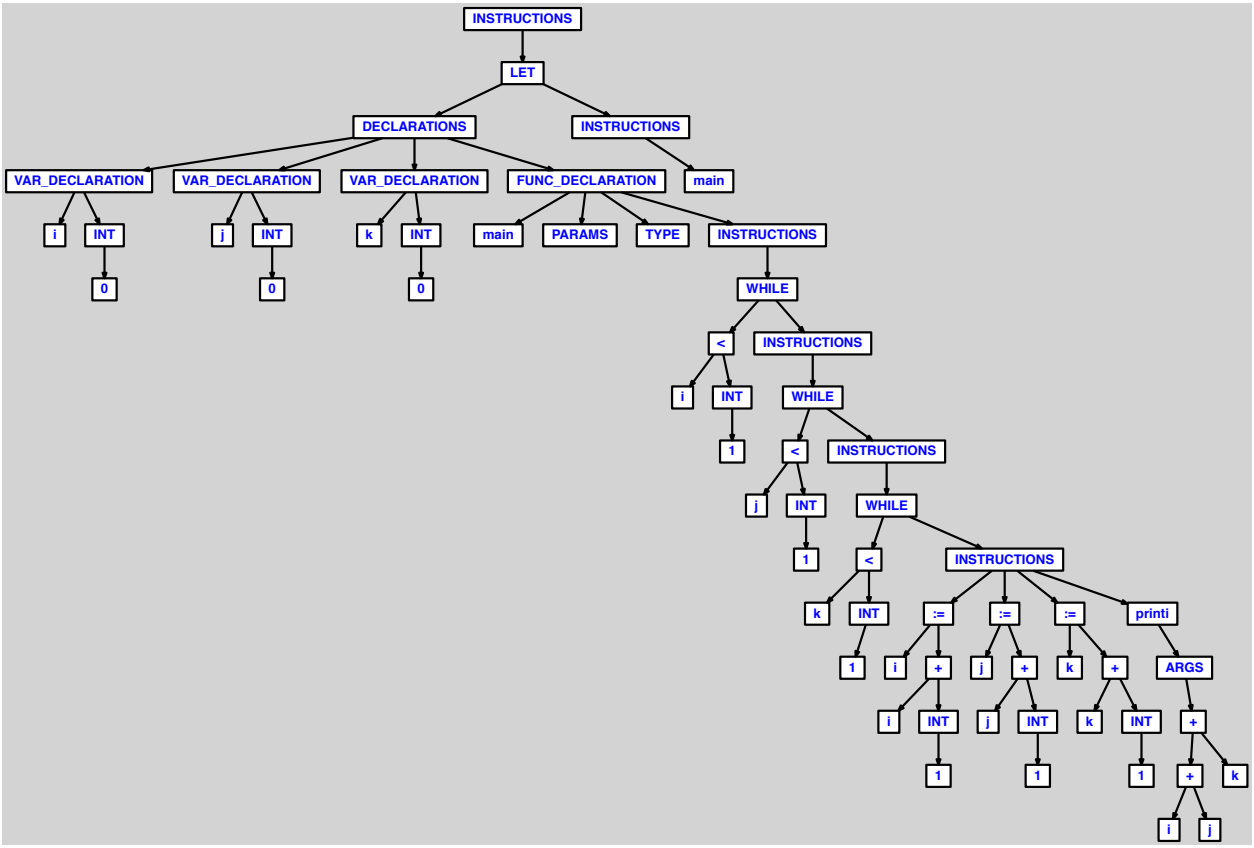
13.2.8 <while> avec double-imbrication

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
main
i
INT
j
INT
main
PARAMS
TYPE
INSTRUCTIONS
0
0
WHILE
<
INSTRUCTIONS
i
INT
WHILE
1
<
INSTRUCTIONS
j
INT
:=
:=
printi
1
i
+
j
+
ARGS
i
INT
j
INT
+
1
1
i
j

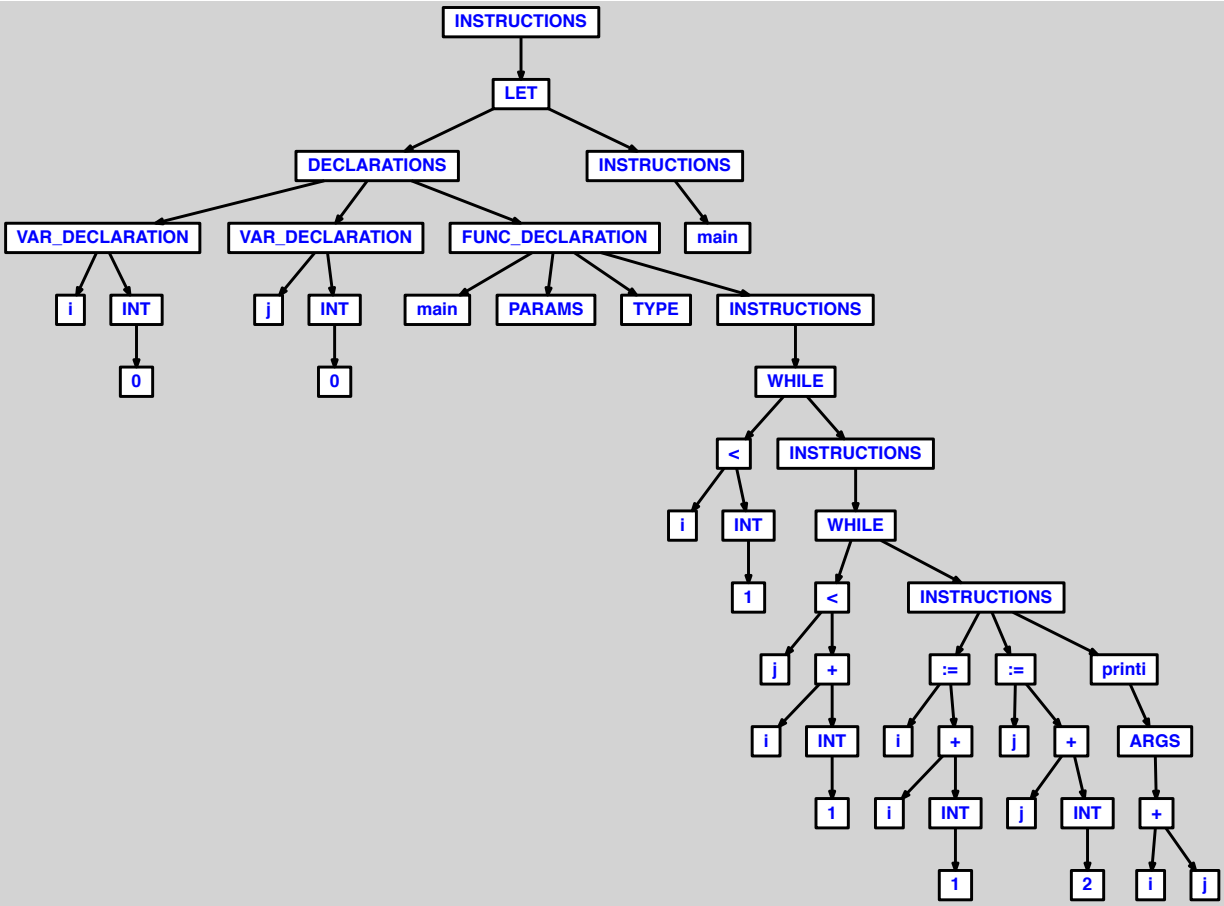
13.2.9 <while> avec triple-imbrication

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
main
i
INT
j
INT
k
INT
main
PARAMS
TYPE
INSTRUCTIONS
0
0
0
WHILE
<
INSTRUCTIONS
i
INT
WHILE
1
<
INSTRUCTIONS
j
INT
WHILE
1
<
INSTRUCTIONS
k
INT
:=
:=
:=
printi
1
i
+
j
+
k
+
ARGS
i
INT
j
INT
k
INT
+
1
1
1
+
k
i
j

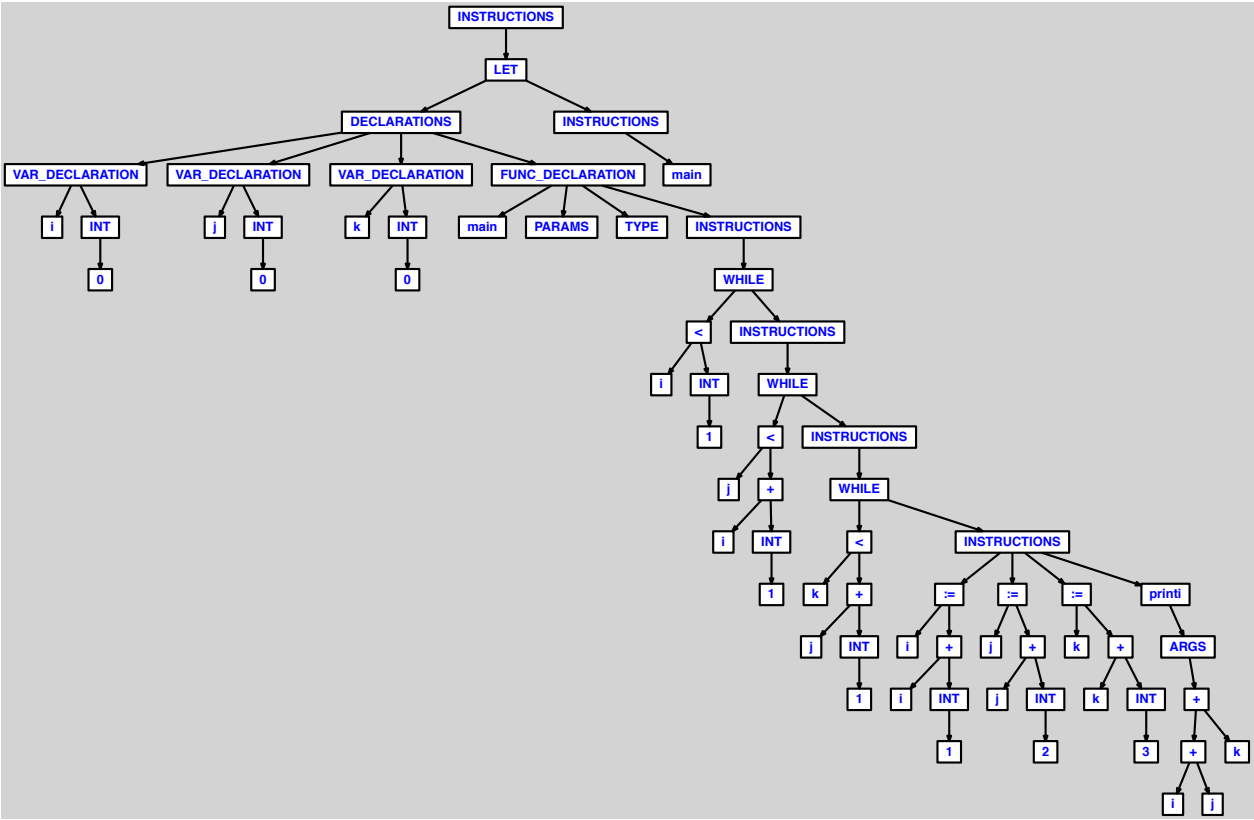
13.2.10 <while> avec reutilisation de compteur

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
main
i
INT
j
INT
main
PARAMS
TYPE
INSTRUCTIONS
0
0
WHILE
<
INSTRUCTIONS
i
INT
WHILE
1
<
INSTRUCTIONS
j
+
:=
:=
printi
i
INT
i
+
j
+
ARGS
1
i
INT
j
INT
+
1
2
i
j

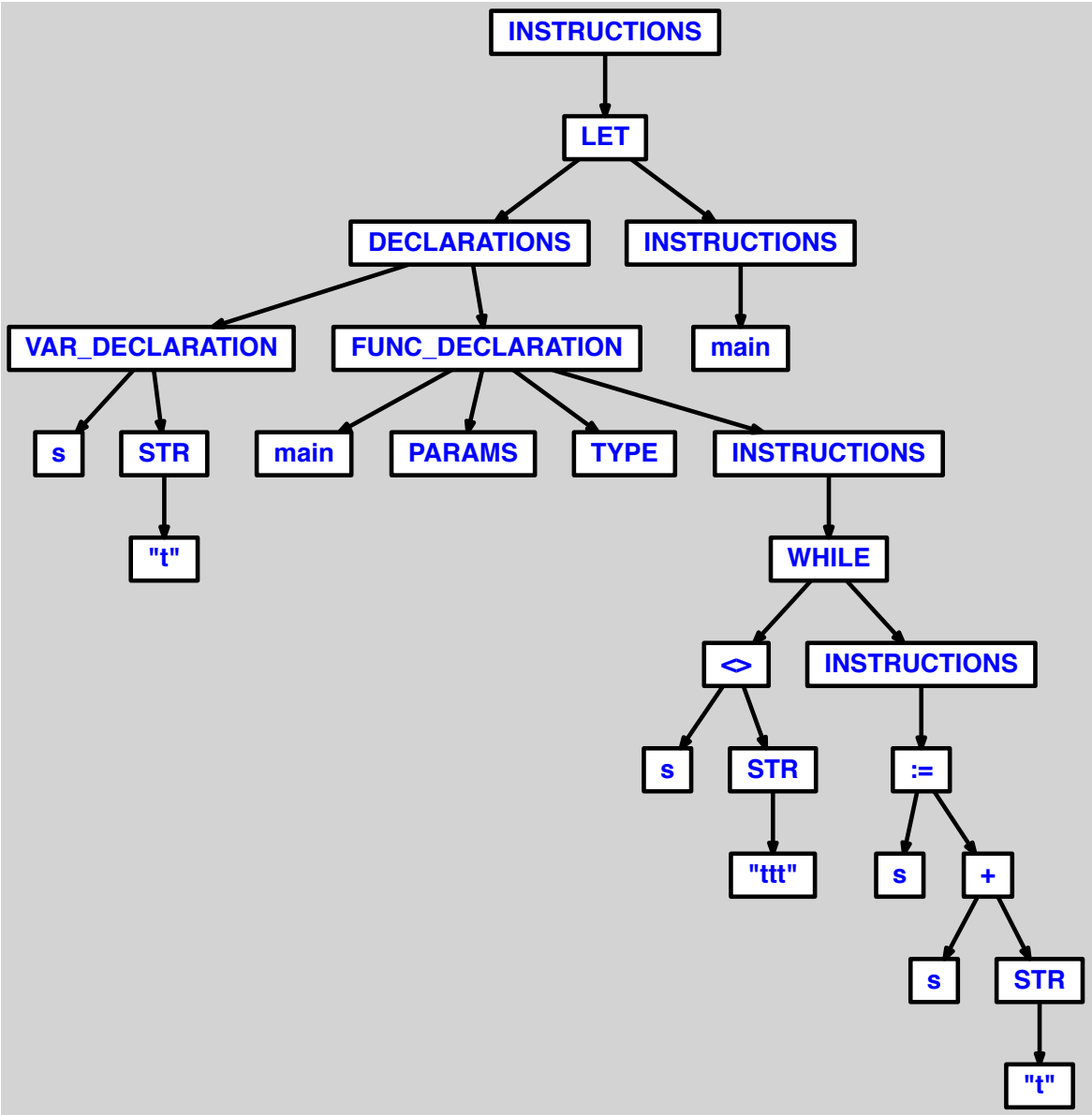
13.2.11 <while> avec reutilisation de compteurs

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
main
i
INT
0
j
INT
0
k
INT
0
main
PARAMS
TYPE
INSTRUCTIONS
WHILE
<
INSTRUCTIONS
i
INT
1
WHILE
<
INSTRUCTIONS
j
+
i
INT
1
WHILE
<
INSTRUCTIONS
k
+
j
INT
1
:=
:=
:=
printi
i
+
i
INT
1
j
+
j
INT
2
k
+
k
INT
3
ARGS
+
+
k
i
j

13.2.12 <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
main
s
STR
main
PARAMS
TYPE
INSTRUCTIONS
"t"
WHILE
<>
INSTRUCTIONS
s
STR
:=
"ttt"
s
+
s
STR
"t"