



Théorie de langage de programmation

Rapport du projet : LAC

Nom & Prénom : Yutong LI

Prénom français David

Date : 2021/1/3

SHANGHAI JIAOTONG UNIVERSITY
SJTU-ParisTech Elite Institute of Technology

Table des matières

1	Introduction	4
1.1	Environnement	4
1.2	Contenu du projet	5
1.3	Mode de compilation	7
2	Analyse Lexicale	8
2.1	Le dessin des expression régulières	8
2.2	L'algorithme pour faire l'analyse lexicale	8
2.3	projet1.c	8
3	Analyse Syntaxique : fonction calculate	10
3.1	Analyse lexicale de l'expression	11
3.2	Définition de grammaire BNF	11
3.3	Automate à pile	11
3.4	Construction de l'arbre syntaxique	12
3.5	Fonction : int calculate_exp_int(const char *, int, int *) {}	12
3.6	projet2.c	12
4	Table des symboles	14
4.1	projet3.c	14
5	Table VM	16
5.1	projet4.c	16
5.2	Chaîne de mots	19
6	Analyse Sémantique : l'exécution du LAC	20
6.1	Les fonctions Conditionnelles	21
6.2	Fonction récursive	22
7	Travaux supplémentaires	23
7.1	Fonction defer	23
7.2	Les variables	23
7.3	Fonction boucle	24
7.4	Vecteur	25
7.5	L'importation	26
7.6	Comparateur < et >	26
7.7	Fuite de mémoire	27
8	Jeu de LAC	28
8.1	factorielle.lac	28
8.2	catenate.lac	30
8.3	fibonacci.lac	30
8.4	sort.lac	31
8.5	strlen.lac	32
9	LAC orienté-objet	34
9.1	Modification de table de symboles	34
9.2	Les objets	35

9.3 LAC <code>class</code>	36
9.4 Gestion des objets (Garbage collection)	37
10 Conclusion	37
11 Appendix	39
11.1 Exécution log du projet4.lac	39
11.2 Exécution log du factorielle.lac	41

1 Introduction

1.1 Environnement

`gcc 7.5.0 (Installé au : /usr/bin/gcc) avec Ubuntu 18.04 LTS`

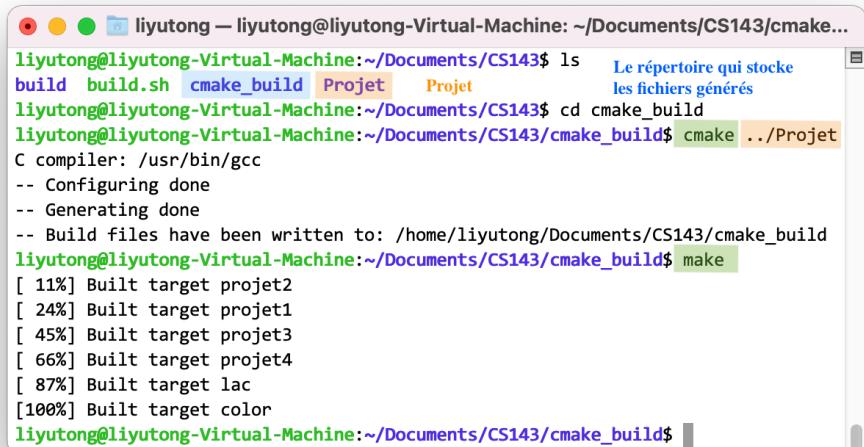
Nous avons fait un projet **cmake**. Il faut que vous utiliser **cmake** pour la compilation du projet. Il faut exécuter les instructions dans le terminal :

```
1 $ mkdir cmake_build && cd cmake_build
2 $ cmake -DCMAKE_BUILD_TYPE=Debug ${PATH_TO_PROJECT}
3 $ make
```

ou le script `build.sh` inclus. Le script demandera si le mode **Debug** est activé :

```
1 $ sudo chmod +x ./build.sh
2 $ export PROJET_DIR=${PATH_TO_PROJECT}
3 $ ./build.sh
4 [ Info ] Enable Debug Mode? [Y/n]
```

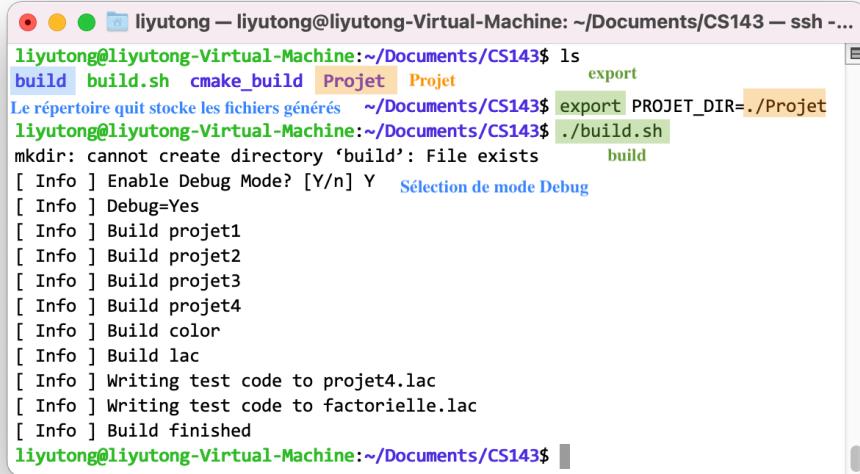
Remarque Il faut que la variable `PROJET_DIR` soit correctement défini (Figure 2 et 1)



The screenshot shows a terminal window with the following session:

```
liyutong@liyutong-Virtual-Machine:~/Documents/CS143/cmake...$ ls      Le répertoire qui stocke
build build.sh cmake_build Projet Projet les fichiers générés
liyutong@liyutong-Virtual-Machine:~/Documents/CS143$ cd cmake_build
liyutong@liyutong-Virtual-Machine:~/Documents/CS143/cmake_build$ cmake ..../Projet
C compiler: /usr/bin/gcc
-- Configuring done
-- Generating done
-- Build files have been written to: /home/liyutong/Documents/CS143/cmake_build
liyutong@liyutong-Virtual-Machine:~/Documents/CS143/cmake_build$ make
[ 11%] Built target projet2
[ 24%] Built target projet1
[ 45%] Built target projet3
[ 66%] Built target projet4
[ 87%] Built target lac
[100%] Built target color
liyutong@liyutong-Virtual-Machine:~/Documents/CS143/cmake_build$
```

FIGURE 1 Instruction de compilation : version cmake



```
liyutong — liyutong@liyutong-Virtual-Machine: ~/Documents/CS143 — ssh -...
liyutong@liyutong-Virtual-Machine:~/Documents/CS143$ ls
build build.sh cmake_build Projet Projet export
Le répertoire quit stocke les fichiers générés ~/Documents/CS143$ export PROJET_DIR=../Projet
liyutong@liyutong-Virtual-Machine:~/Documents/CS143$ ./build.sh
mkdir: cannot create directory 'build': File exists build
[ Info ] Enable Debug Mode? [Y/n] Y Sélection de mode Debug
[ Info ] Debug=Yes
[ Info ] Build projet1
[ Info ] Build projet2
[ Info ] Build projet3
[ Info ] Build projet4
[ Info ] Build color
[ Info ] Build lac
[ Info ] Writing test code to projet4.lac
[ Info ] Writing test code to factorielle.lac
[ Info ] Build finished
liyutong@liyutong-Virtual-Machine:~/Documents/CS143$
```

FIGURE 2 Instruction de compilation : version build.sh

Les commandes gèneront des exécutables dans ./build/ :

- **projet1** – **projet4** Programmes pour projet 1-4
- **lac** L'exécuteur lac complet (projet 5)
- **color** Visualisation du surligneur de code

1.2 Contenue du projet

Le répertoire de projet. Par example, le **Projet-release** est présenté dans la Figure 3 :

```

CMakeLists.txt
└── common
    ├── btree.h / btree.c
    ├── macros.h
    ├── queue.h / queue.c
    ├── stack.h / stack.c
    └── types.h
└── demo
    ├── projet1-analex.c
    ├── projet2-calculate.c
    ├── projet3-symtable.c
    └── projet4-lac.c
└── lac.c
└── lex
    └── analex.h / analex.c
└── playground
    ├── boucle.lac
    ├── catenate.lac
    ├── defer.lac
    ├── factorielle.lac
    ├── fibonacci.lac
    ├── library.lac
    ├── memchk.lac
    └── sort.lac
└── runtime
    ├── env.h / env.c
    ├── function.h / function.c
    ├── interpret.h / interpret.c
    ├── proc.h / proc.c
    ├── symtable.h / symtable.c
    └── vmtable.h / vmtable.c
└── tests
    └── ...
└── tools
    ├── build.sh
    ├── color.c
    └── memchk.sh
└── utils
    ├── calculate.h / calculate.c
    ├── convert.h / convert.c
    ├── debug.h / debug.c
    └── io.h / io.c

```

FIGURE 3 Contenu du Projet-release

Les fonctionnement de chaque fichier est expliqué dans leurs propres commentaires :

— **common** Composantes communes

- **demo** Programmes pour projet
- **lex** Analyse lexicale
- **runtime** Fonctions nécessaires à l'exécution du lac
- **tests** Programmes pour essai
- **tests** Les exemples de LAC
- **utils** Fonction instrumentale

1.3 Mode de compilation

Souvent, nous voulons que le programme nous renvoie des informations importantes pour déboguer. Si le Projet est compilé avec :

```
1 $ cmake -DCMAKE_BUILD_TYPE=Debug ${PATH_TO_PROJECT}$
```

Alors les informations pour déboguer seront déposés par les programmes dans le terminal (Figure 4) :

```

● ○ ⊞ ilyutong@ilyutong-Virtual-Machine: ~/Documents/CS143/cmake_build ── ssh -p 60001 ilyutong@59.78.22.189 - 133x57
ilyutong@ilyutong-Virtual-Machine:~/Documents/CS143/cmake_build$ ./lac ..../Projet/tests/projet4.lac
----- VM Table -----
0:-1 1:0 2:-1 3:1 4:-1 5:2 6:-1 7:3 8:-1 9:4 10:-1
11:5 12:-1 13:6 14:-1 15:7 16:-1 17:8 18:-1 19:9 20:-1
21:10 22:-1 23:11 24:-1 25:12 26:-3 27:13 28:-3 29:14 30:-3
31:15 32:-3 33:16 34:-3 35:17 36:-3 37:18 38:-3 39:19 40:-3
41:20 42:-3 43:21
-----
----- Symbol Table -----
dup(9) -> drop(2) -> swap(4) -> .(6) -> count(8) -> type(10) -> +(12) -> *(14) -> +(16) -> -(18) -> (lit)(20) -> (fin)(22) -> calculate(24) -> if(26) -> else(28) -> then(30) -> recurse(32) -> @!(34) -> !(36) -> while(38) -> loop(40) -> break(42)
-----
----- Assembly result -----
M("variable")->M("x")->N("2")->M("1")->M("test")->M("while")->M("x")->M("0")->M("=")->M("if")->M("break")->M("else")->M("x")->M("0")->N("1")->M("x")->M("1")->M("then")->M("x")->M("0")->M("loop")->M("1")->M("test")->
-----
----- Highlight result -----
variable x
2 x i
: test while x @ 0 = if break else x @ 1 - x ! then x @ . loop ;
test
-----
[ Debug ] A variable x is defined at VMTable[44]
[ Debug ] Push 2 into StkData
[ Debug ] Execute lac function at VMTable[44]
[ Debug ] Push 45 into StkReturn
[ Debug ] Execute basic function (lit) at VMTable[20]
[ Debug ] Pop 45 from StkReturn, StkReturn.top = N/A
[ Debug ] Push 46 into StkReturn
[ Debug ] Push 48 into StkData
[ Debug ] Pop 46 from StkReturn, StkReturn.top = N/A
[ Debug ] Push 47 into StkReturn
[ Debug ] Execute VMTable[47]
[ Debug ] Execute basic function (fin) at VMTable[22]
```

FIGURE 4 Mode Debug du lac

Si le Projet est compilé avec :

```
1 $ cmake -DCMAKE_BUILD_TYPE=Release ${PATH_TO_PROJECT}$
```

Alors il y n'aura pas de informations supplémentaire.

2 Analyse Lexicale

2.1 Le dessin des expression régulières

Dans cette étape, nous faisons l'analyse lexicale avec les expressions régulières. Les expressions régulières que nous avons choisi pour la détection des lexèmes sont présenté dans Tableau 1 :

Lexème	Expression	Note
Commentaires	<code>\n \s][\\]*?\) [\n \s]\\[^\\n]*</code>	
Chaînes de mots	<code>\n \s]"[^"]*?"</code>	
Identificateurs	<code>[^\\s\\n]+</code>	
Nombres entières	<code>(\.- \\+)?[0-9]+(\\.?[0-9]+)?[\\s \\n]</code>	Ceci est en fait l'expression pour les nombres réelles

TABLE 1 Les expressions régulières

Actions additionnels : L'interpréteur ajout sauts de ligne **au début** et **à la fin** des caractères entrées en autonome.

2.2 L'algorithme pour faire l'analyse lexicale

1. Scanner la chaîne de saisie.
2. Commencer par discerner une commentaire
3. Discerner une chaîne de caractères entre la tête de la chaîne d'entrée et la tête du commentaire.
4. Discerner les identificateurs entre la tête de la chaîne d'entrée et la tête de la chaîne de caractères.
5. Discerner la chaîne suivante de caractères, puis discerner l'identifiant entre la fin de la chaîne précédente et la tête de la chaîne suivante, répétez jusqu'à la tête de la commentaire.
6. Discerner le commentaire suivante, répétez.

Cette algorithme est réalisé dans le fichier `lex/analex.c`.

2.3 projet1.c

Le fichier `projet1.c` (voir Figure 5) est simple.

```

1 #include <stdio.h>
2 #include "../common/queue.h"
3 #include "../lex/analex.h"
4 #include "../utils/debug.h"
5 #include "../utils/io.h"
6
7 int main(int argc, const char *argv[]) {
8     char *psReadBuffer = NULL;
9     FILE *pInputFile ;
10
11    /* Usage : ${PATH_TO_EXE}/${EXE} ${FILENAME}
12     * Usage : cat ${FILENAME} | ${PATH_TO_EXE}/${EXE}
13     */
14
15    /* redirect input to pInputFile pointer*/
16    assign_input_stream(argc, argv, &pInputFile);
17
18    long ulReadCnt = input_to_buffer(pInputFile, &psReadBuffer);
19    if (ulReadCnt ≤ 0) exit(ERR_IO); // Failed to read any bytes
20
21    /* Queue where the lexemes will be stored */
22    queue_t queRes ;
23    queue_init(&queRes) ;
24
25    /* Perform Analyze lexical */
26    match_lac(psReadBuffer, psReadBuffer + ulReadCnt, &queRes) ;
27
28    /* Visualization of results */
29    disp_analex_res(psReadBuffer, &queRes) ;
30    visualize(psReadBuffer, &queRes) ;
31    fflush(stdout) ;
32
33    /* Free memory */
34    queue_clear(&queRes) ;
35    return 0 ;
36 }

```

FIGURE 5 projet1.c

La fonction de fiche est bien expliquée par les commentaires. Nous le exécutons dans le terminal de manière suivant

```
1 ./projet1_factorielle.lac
```

Le programme fera l'analyse lexicale, plus il renvoie les résultats. Nous utilisons quelques astuces pour produire du texte souligné en couleur sous un terminal Linux.

```

liyutong@liyutong-Virtual-Machine:~/Documents/CS143/cmake_build$ ./projet1 ..//Projet/tests/factorielle.lac
---- Annalex result ----
S("uu")->M("fact")->M("dup")->M("0=")->M("if")->M("drop")->N("1")->M("else")->M("dup")->M("1-")->M(
"reurse")->M("*")->M("then")->M(";")->M("go")->S("Factorielle")->M("count")->M("type")->M("dup")->
M(",")->S("vaut :
")->M("count")->M("type")->M("fact")->M(".")->M("cr")->M(";")->N("6")->M("go")->
-----
----- Highlight result -----
\ Fichier " factorielle.lac"
" uu"
( Ce fichier est un "exemple" étudié pour tester
l'analyseur lexical écrit en phase 1 du projet)
\ BlahBlahBlah
: fact ( n -- n!)
    dup 0=
    if
        drop 1
    else
        dup 1- recurse *
    then ;

: go ( n -- )
    " Factorielle" count type
    dup .
    " vaut :
" count type
    fact . cr ;
6 go
-----
liyutong@liyutong-Virtual-Machine:~/Documents/CS143/cmake_build$ 

```

FIGURE 6 Exportation du programme `projet1`

3 Analyse Syntaxique : fonction calculate

Nous développons une fonction `calculate_exp_int` pour la génération d'une arbre syntaxique. Il y a trois étapes :

1. Faire l'analyse lexicale. L'expression est une chaîne de caractère. Il faut que nous le transformons comme une chaîne de lexèmes.
2. Avec un automate des piles, transformer l'expression infixe au expression postfixée.
3. Générer un arbre grammaire à partir de l'expression postfixée. Évaluer l'expression en même temps.

Nous écrivons une fonction `calculate_exp` pour faire à la fois des calculs des expressions et des analyses syntaxiques. Les trois étapes sont réalisé par :

```

— decode_exp(&queSymbols, sExpression, iNumBytes);
— gen_postfix_exp(&quePostfixExp, &queSymbols);
— build_exp_tree(&quePostfixExp, &ExpTreeRoot);

```

3.1 Analyse lexicale de l'expression

Nous avons déjà fait l'analyse lexicale dans projet1 avec les expressions régulières. Donc pour la première nous avons créé un automate finis déterministe. Ce n'est pas intéressant à expliquer(voir la fonction `decode_exp(queue_t *, char *, int)` du fichier `utils/calculate.c`)

3.2 Définition de grammaire BNF

Pour la deuxième étape, nous développons la grammaire BNF des expressions.

```

< chiffre > ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
< naturel > ::= < chiffre > < chiffre >
< op_lp > ::= "+"|"-"|"*"
< op_hp > ::= "x"|"/"
< op_sgn > ::= "+"|"-"|"*"
< entier > ::= < op_sgn >< naturel >
< op > ::= < op_lp > | < op_hp > | < op_sgn >
< facteur > ::= < entier > | < facteur >< op_hp >< entier > | "(" < terme > ")"
< terme > ::= < facteur > | < terme >< op_lp >< facteur >
< somme > ::= < entier > | < somme > "+" < entier >
    
```

Explication : 1) Nous permettons les expression comme `1+3`. 2) Nous permettons la présence de `/` comme la division. 3) Les nombre réelles sont interprète comme entiers.

3.3 Automate à pile

Nous réalisons un automate des piles de Figure 7 par `gen_postfix_exp(queue_t *, queue_t *)`. Cet automate serve à translater une expression infixée à une expression postfixée.

Interpretation

- Quand nous rencontrons les nombres (**N**), nous les posons dans le résultat (**res**).
- Si nous rencontrons les `'('`, nous les posons dans le pile de opérateurs (**P**)
- Si nous rencontrons les `')'`, nous sommes en état 1. Nous déplions les opérateur de **P** dans **res** jusqu'à `'('` ou vide. Nous enlevons le `'('`
- En cas d'un opérateur. Si le pile est vide, nous empilons le opérateur dans **P**. Sinon, nous déplions les opérateurs qui sont supérieurs que ce opérateur dans **res** et nous le empilons dans **P**

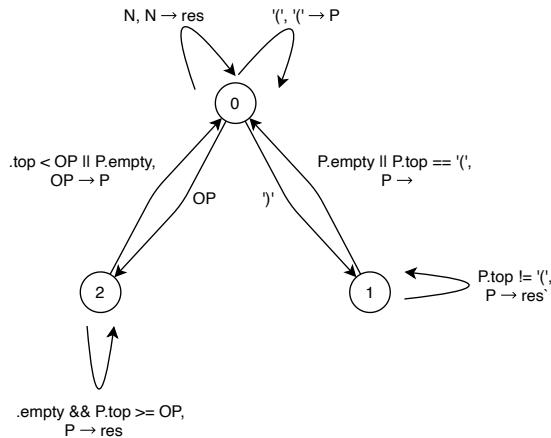


FIGURE 7 Schéma de l'automate

3.4 Construction de l'arbre syntaxique

Finalement, nous générerons un arbre grammaire à l'aide de la fonction `build_exp_tree(queue_t *, exp_btnode_t *)`. L'algorithme est simple :

1. Si nous rencontrons les nombre (N), nous les empilons comment les noeud.
2. Si nous rencontrons les OPs, nous dépiles deux noeud, et les connecté sur l'OP. Nous empilons l'OP.
3. Nous répétons l'étape 1 et 2, jusqu'à il y a seulement un noeud dans le pile. Ceci est le noeud principal de l'arbre.

3.5 Fonction : int calculate_exp_int(const char *, int, int *) {}

Les codes de ces fonctions sont inclus dans `utils/calculate.c`. Ils sont bien expliqués par leurs propres commentaires.

3.6 projet2.c

Nous écrivons une program `projet2.c` pour tester la fonction `calculate_exp_int` :

```

1 #include "../utils/calculate.h"
2 #define EXP_MAX_LEN 0x7F
3
4 int main(int argc, const char *argv[]) {
5     char *sExp;
6     int res;
7
8     if (argc > 1) {
9         /* Use input parameter as expression */
10        sExp = (char *) argv[1];
11    } else {

```

```

12     sExp = (char *) calloc(sizeof(char), (EXP_MAX_LEN + 1));
13     if (sExp == NULL) exit(ERR_MEM);
14     printf("Input expression: ");
15     if (fgets(sExp, EXP_MAX_LEN, stdin) == NULL) exit(ERR_IO);
16 }
17
18 int iNumBytes = (int) strlen(sExp);
19 calculate_exp_int(sExp, iNumBytes, &res);
20 printf("Result: %d\n", res);
21 return 0;
22 }
```

L'expression est calculé, un arbre syntaxique est généré (voir Figure 8).

```

liyutong — liyutong@liyutong-Virtual-Machine: ~/Documents/CS143/cmake_build...
liyutong@liyutong-Virtual-Machine:~/Documents/CS143/cmake_build$ ./projeto2
Input expression: -(1-2)+(3-4)x(-5)
-(1-2)+(3-4)x(-5)

Calculate expression: -(1-2)+(3-4)x(-5)

--- Postfix expresion ---
1 2 - - 3 4 - 0 5 - x +
-----
----- Tree -----
+
-
x
2
1   2   3   4   0   5
-----
[ info ] Expression value: 6
Result: 6
liyutong@liyutong-Virtual-Machine:~/Documents/CS143/cmake_build$ 
```

FIGURE 8 Exportation du `projeto2.c`

Remarque : Il faut compiler le projet en mode **Debug** pour voir la visualisation. Visualisation des arbres est difficile. Les 0 sont ajouter avant les signes pour qu'ils peuvent être calculé comme les '+' et les '-'.

4 Table des symboles

Cette partie est simple : nous créons la structure de donnée `symtable_t` pour stocker les symboles. Puis nous faisons ce que nous avons étudié et discuté dans la lecture. La liste des fonctions qui nous permet l'interaction avec un table des symboles est présente dans Figure 9

```

1  /* Init the table */
2  int symtable_init(symtable_t *pSymTable) ;
3
4  /* Destroy a table */
5  int symtable_destroy(symtable_t *pSymTable) ;
6
7  /* Clear a table */
8  int symtable_clear(symtable_t *pSymTable) ;
9
10 /* Compare the symbol with an extraction of symbol */
11 int symtable_cmp(u_symbol *pSymTableLoc, u_symbol *pInput) ;
12
13 /* Search the symbol in table */
14 int symtable_search(symtable_t *pTable, u_symbol *pInput) ;
15
16 /* Expand the table */
17 int symtable_expand(symtable_t *pSymTable) ;
18
19 /* Add symbol to table */
20 int symtable_add(symtable_t *pSymTable, const char *pSymbolStr, int iLength) ;
21
22 /* Get CFA of a symbol by its idx in the table */
23 int symtable_get_cfa(symtable_t *pSymTable, int iSymbolIdx) ;
24
25 /* Set CFA of a symbol by its idx in the table */
26 int symtable_set_cfa(symtable_t *pSymTable, int iSymbolIdx, int iCFAVal) ;

```

FIGURE 9 Une liste des fonctions qui interagir avec le table de symboles

4.1 projet3.c

Le code qui teste la fonctionnement du `symtable_t` est présenté dans Figure 10

```

1 #include "../runtime/symtable.h"
2 #include "../utils/debug.h"
3
4 int main() {
5
6     symtable_t SymTable;
7     symtable_init(&SymTable);
8     symtable_add(&SymTable, "+", 1);
9     symtable_add(&SymTable, "swap", 4);
10    symtable_add(&SymTable, "swap", 4);
11    symtable_add(&SymTable, "swapp", 5);
12    disp_symtable(&SymTable);
13
14    return 0;
15 }
16
17 }
```

FIGURE 10 projet3.c

Les exportations sont présenté dans la Figure 11

```

liyutong — liyutong@liyutong-Virtual-Machine: ~/Documents/CS143/build...
liyutong@liyutong-Virtual-Machine:~/Documents/CS143/build$ ./projet3
[ Warning ] Duplicated symbol found: swap
----- Symbol Table -----
+(0) -> swap(0) -> swapp(0)
-----

liyutong@liyutong-Virtual-Machine:~/Documents/CS143/build$
```

FIGURE 11 Exportation du projet3.c

Interprétation : Nous voyons que la deuxième symbole `swap` est éliminé. Il y a un "Warning" qui indique que la deuxième lexème `swap` est redondante. C'est une redéfinition de fonctions `swap`. Le programme peut bien chercher les symboles dans le table et ajouter les symboles. Les nombre après les symboles sont leurs CFA. Comme nous n'avons pas crée le table VM, leurs valeurs sont tous nul.

5 Table VM

Dans cette section, nous réalisons une liste des fonctions qui nous permettent de computer la langage du LAC. Ils sont :

- `interpret` qui fonctionnent dans mode interprétation.
- `compile_function` qui fonctionnent dans mode de compilation.
- `exec_vm` qui fonctionnent dans mode d'exécution.
- les fonctions de base

Pour rendre la logique plus claire, nous avons crée une variable global `g_env` qui represent l'environnement :

```

1  typedef struct vmenv_t {
2      stack_t StkData;           // Data stack
3      stack_t StkReturn;        // Return stack
4      symtable_t SymTable;     // Table of symbols
5      vmtable_t VMTTable;      // VM table
6      strtable_t StrTable;     // String table
7      bool bCompiled;          // Flag
8      bool bInitiated;         // Flag
9  } vmenv_t;
10
11 vmenv_t g_Env = {0};

```

Comme discuté dans le cours, nous avons codé les fonctions de base avec un -1 et les fonctions LAC avec un -2 pour évité les conflits. En plus de cela, nous suivons les exigences pour rendre le table VM.

5.1 projet4.c

Donc le projet4.c (Dans Figure 12) font 2 chose :

1. Analyse lexicale de fichier `factorielle.lac`
2. L'interprétation avec analyse sémantique (et analyse syntaxique si nous voulons réaliser `defer`)

```

1 #include "../common/types.h"
2 #include "../lex/analex.h"
3 #include "../runtime/interpret.h"
4
5 int main() {
6     extern vmenv_t g_Env;
7     g_proc_env_init();
8     g_proc_compile();
9     disp_vmtable(&g_Env.VMTable);
10    disp_symtable(&g_Env.SymTable);
11
12    char *psReadBuffer = NULL;
13    FILE *pInputFile;
14    pInputFile = fopen("projet4.lac", "r");
15    int iReadCnt = input_to_buffer(pInputFile, &psReadBuffer);
16
17    queue_t queRes;
18    queue_init(&queRes);
19    match_lac(psReadBuffer, psReadBuffer + iReadCnt, &queRes);
20    disp_analex_res(psReadBuffer, &queRes);
21    visualize(psReadBuffer, &queRes);
22    fflush(stdout);
23
24    interpret(psReadBuffer, &queRes);
25
26
27    return 0;
28
29 }

```

FIGURE 12 projet4.c

Nous avons déjà fait l'analyse lexicale dans `projet1`. En revanche, l'interprétation est fait par les fonctions `interpret`, `compile_function` et `exec_vm`

Nous lançons d'abord la fonction `void interpret(char *psReadBuffer, queue_t *pqueRes)` avec `psReadBuffer` les caractères du LAC et `pqueRes` une queue des symboles. Si le symbole est un nombre, il est empilé dans le pile de données. Si le symbole est une chaîne de mots, il est stocké dans une structure `strtable_t` et son `address(int)` est empilé dans le pile de données.

Si le symbole est un identificateur, la fonction `interpret` vérifiera l'existence de cette symbole. Si oui, nous lançons la fonction `void exec_vm(int iCFA)`. Sinon, il y a une erreur sémantique.

Il existe des **mots clés**, e.g. : et ;, qui marquent le début et la fin d'une fonction LAC. Ils sont procédé par `void compile_function(char *psReadBuffer, queue_t *pqueRes)`

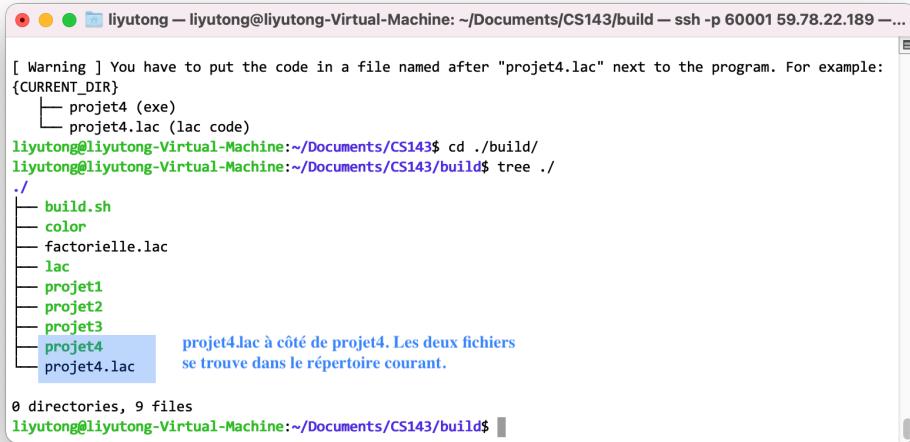
Nous avons testé notre programme avec l'exemple donné sur le moodle :

```

1 : incr 1 +
2
3 : 2+. incr incr . ;
4
5 7 8 + 2+.

```

Nous mettons le code LAC dans le fichier `projet4.lac` à côté de l'exécutable `projet4` dans le répertoire courant.. Si le programme ne trouve pas le fichier du LAC, il renverra une instructions à l'utilisateur pour mettre le fichier correctement. (Figure 13)



The screenshot shows a terminal window with the following content:

```
[ Warning ] You have to put the code in a file named after "projet4.lac" next to the program. For example:  
{CURRENT_DIR}  
└── projet4 (exe)  
    └── projet4.lac (lac code)  
liyutong@liyutong-Virtual-Machine:~/Documents/CS143$ cd ./build/  
liyutong@liyutong-Virtual-Machine:~/Documents/CS143/build$ tree ./  
./  
├── build.sh  
├── color  
├── factorielle.lac  
├── lac  
├── projet1  
├── projet2  
├── projet3  
└── projet4  
    └── projet4.lac  
        projet4.lac à côté de projet4. Les deux fichiers  
        se trouvent dans le répertoire courant.  
0 directories, 9 files
```

FIGURE 13 Comment exécuter le projet4

Puis, nous exécutons dans le terminal.

```
1 $ ./projet4
```

Les résultat sont présentés dans Figure 14. En plus, dans section11.1 se trouve la version détaillé du log de l'exécution de `projet4.c` (Si le projet est compilé en mode **Debug**, voir Section 1.3).

```

liutong@liutong-Virtual-Machine: ~/Documents/CS143/cmake_build -- ssh -p 60001 liutong@59.78.22.18...
liutong@liutong-Virtual-Machine: ~/Documents/CS143/cmake_build$ vim projet4.lac
liutong@liutong-Virtual-Machine: ~/Documents/CS143/cmake_build$ ./projet4
----- VM Table -----
0:-1 1:0 2:-1 3:1 4:-1 5:2 6:-1 7:3 8:-1 9:4 10:-1
11:5 12:-1 13:6 14:-1 15:7 16:-1 17:8 18:-1 19:9 20:-1
21:10 22:-1 23:11 24:-1 25:12 26:-3 27:13 28:-3 29:14 30:-3
31:15 32:-3 33:16 34:-3 35:17 36:-3 37:18 38:-3 39:19 40:-3
41:20 42:-3 43:21
-----
----- Symbol Table -----
dup(0) -> drop(2) -> swap(4) -> .(6) -> count(8) -> type(10) -> =(12) -> *(14) -> +(16) -> -(18) -> (lit)(20) ->
(fin)(22) -> calculate(24) -> if(26) -> else(28) -> then(30) -> recurse(32) -> @!(34) -> !(36) -> while(38) -> loop(40) -> break(42)
-----
----- Annalex result -----
M(";) -> M("incr") -> N("1") -> M("+-") -> M("::") -> M("2+.") -> M("incr") -> M(".") -> M(";-") -> N("7") -> N("8")
-> M("+-") -> M("2+.") ->
-----
----- Highlight result -----
: incr 1 + ;
: 2+. incr incr . ;
7 8 + 2+.
-----
[ Debug ] Define function incr with CFA = 44
[ Debug ] Define function 2+. with CFA = 49
[ Debug ] Push 7 into StkData
[ Debug ] Push 8 into StkData
[ Debug ] Execute basic function + at VMTTable[16]
[ Debug ] Pop 8 from StkData, StkData.top = 7
[ Debug ] Pop 7 from StkData, StkData.top = N/A

```

FIGURE 14 Exportation du projet4.c

5.2 Chaîne de mots

Nous stockons les chaînes de mots dans le table de VM (Voir Tableau 2) pour stocker les chaînes de mots. Une chaîne de mots peut être représenté par son Numéro dans **strtable_t**. Les chaînes sont séparées par des 0.

Numéro	x-1	x	x+1	x+2
VM	...	72	105	0
Signification	...	H	i	séparateur

TABLE 2 Schéma du table de VM

Interprétation Dans cet exemple, la chaîne de mots "Alain" peut être présenté par *x*. Nous pouvons visiter cette chaîne par `g_Env.VMTable[x]`.

En mode interpréter, nous posons la chaîne que nous rencontrons ("Alain") dans le table de VM et nous posons sont adresse(*x*) sur le pile de données.

En mode compilé, c'est plus compliqué. L'insertion brutale d'une chaîne de caractères dans la table VM provoque une erreur d'exécution. Donc nous ajoutons d'abord un fonction de base `jr` (Figure 15) qui saute le chaîne de caractère (même fonctionnement que `else`). Puis nous posons l'address de chaîne après une fonction (`lit`). Tableau 3 est une démonstration.

```

1 void proc_func_jr() {
2     int iAddr = stack_pop_vm(&g_Env.StkReturn);
3     stack_push_vm(&g_Env.StkReturn, g_Env.VMTable.OpCodes[iAddr + 1]);
4 #ifdef DEBUG
5     printf("\n[ Debug ] Unconditional jump to %d from %d\n", ...
6         g_Env.VMTable.OpCodes[iAddr + 1], iAddr);
7     fflush(stdout);
8 }
```

FIGURE 15 Fonction de base `jr`

Numéro	x-1	x	x+1	x+2	x+3	x+4	x+5	x+6
VM	...	JR	x+4	72	105	0	LIT	x+2
Signification	...	CFA de jr	Dst de jr	H	i	sép	CFA de (lit)	Adresse

TABLE 3 Schéma du table de VM en mode compiler

En effet, les chaîne de mots ont des structure très similaire que les vecteurs. Nous les discuterons dans les pages suivants.

6 Analyse Sémantique : l'exécution du LAC

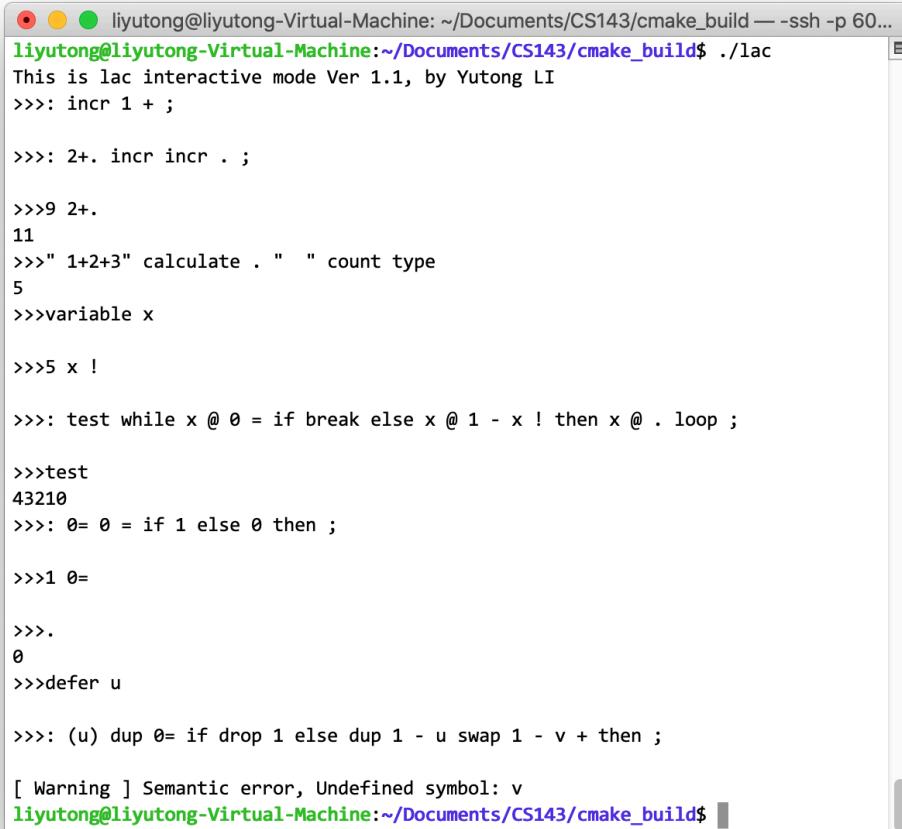
Dans cette partie, nous avons réaliser un vrai exécuteur du LAC (comme ce que du python). Nous pouvons le lancer par :

```
1 $ ./lac ${FILE_LAC}$
```

ou simplement :

```
1 $ ./lac
```

La deuxième méthode active le mode interactif du exécuteur : on peut taper dans le terminal est voir les résultats immédiatement (voir Figure 16). L ‘exécuteur peut détecter les erreurs sémantiques et et les erreurs syntaxiques.



```

liyutong@liyutong-Virtual-Machine: ~/Documents/CS143/cmake_build -- ssh -p 60...
liyutong@liyutong-Virtual-Machine:~/Documents/CS143/cmake_build$ ./lac
This is lac interactive mode Ver 1.1, by Yutong LI
>>>: incr 1 + ;
>>>: 2+. incr incr . ;
>>>9 2+.
11
>>>" 1+2+3" calculate . " " count type
5
>>>variable x
>>>5 x !
>>>: test while x @ 0 = if break else x @ 1 - x ! then x @ . loop ;
>>>test
43210
>>>: 0= 0 = if 1 else 0 then ;
>>>1 0=
>>>.
0
>>>defer u
>>>: (u) dup 0= if drop 1 else dup 1 - u swap 1 - v + then ;
[ Warning ] Semantic error, Undefined symbol: v
liyutong@liyutong-Virtual-Machine:~/Documents/CS143/cmake_build$ 
```

FIGURE 16 Mode interactif

6.1 Les fonctions Conditionnelles

Comme présenté dans la Figure 16, les fonctions conditionnelles marchent bien. Les fonction de base sont :

```

1 void proc_func_if() {
2     int iCond = stack_pop_vm(&g_Env.StkData); // [?] Consumes the top of data stack
3     int iAddr;
4     if (iCond > 0) {
5         iAddr = stack_pop_vm(&g_Env.StkReturn); 
```

```

6         stack_push_vm(&g_Env.StkReturn, iAddr + 1);
7     } else {
8         iAddr = stack_pop_vm(&g_Env.StkReturn);
9         stack_push_vm(&g_Env.StkReturn, g_Env.VMTable.OpCodes[iAddr + 1]);
10    }
11 }
12
13 void proc_func_else() {
14     /* If else symbol is encountered, then the if branch must be taken */
15     /* Jump to the end of conditonal branch */
16 #ifdef DEBUG
17     printf("\n [ Debug ] Function Else\n");
18 #endif
19     proc_func_jr();
20 }
21
22
23 void proc_func_then() {
24     ;
25 }
```

Nous utilisons une pile de `if` et une pile de `else` pour l'analyse syntaxique. L'algorithme s'écrit :

1. Si nous rencontrons `if`, nous empilons son CFA dans le pile de `if`.
2. Si nous rencontrons `else`, nous empilons son CFA dans le pile de `else` et nous dépilons CFA de dernier `if`. Nous modifions la valeur de VM qui correspond à ce CFA (link).
3. Si nous rencontrons `then`, nous dépilons CFA de dernier `if` ou `else` le plus proche. Nous modifions la valeur de VM qui correspond à ce CFA (link).
4. Si tous les piles sont vide quand la compilation est terminé, il n'y a pas de erreur syntaxique.

Cet algorithme est réalisé dans `runtime/interpret.c`

6.2 Fonction récursive

Comme présenté dans Figure 17 Notre exécuteur soutient naturellement les fonctions récursives par le mots clé `recurse` :

```

1 : v ( n -- v[n])
2 dup 0= if drop 1 else dup 1 - u 2 * swap 1 - recurse - then ;
```

ou simplement le nom de la fonction LAC :

```

1 : v ( n -- v[n])
2 dup 0= if drop 1 else dup 1 - u 2 * swap 1 - v - then ;
```

7 Travaux supplémentaires

7.1 Fonction defer

Nous avons réalisé la fonction `defer` dans le mode interprétation (voir Figure 17).

```

liyutong — liyutong@liyutong-Virtual-Machine: ~/Documents/CS143/cmake_build...
>>>liyutong@liyutong-Virtual-Machine:~/Documents/CS143/cmake_build$ ./lac
This is lac interactive mode Ver 1.1, by Yutong LI
>>>defer u \ Déclaration de u

>>>: 0= 0 = ;

>>>: v ( n -- v[n]) dup 0= if drop 1 else dup 1 - u 2 * swap 1 - recurse - then
;

>>>: (u) ( n -- u[n]) dup 0= if drop 1 else dup 1 - u swap 1 - v + then ;
>>>' (u) is u \ Résolution de la déclaration

>>>4 u .
9
>>>

```

FIGURE 17 Fonction `defer`

7.2 Les variables

Les variables sont définies comme une fonction LAC spéciale. Ils sont définis dans le tableau de VM comme la Tableau 4

Numéro	x	x+2	x+3	x+4	x+5
VM	-2	20	x+5	22	?
Signification	FUNC_LAC	(lit)	idx	(fin)	value

TABLE 4 Schéma du tableau de VM pour une variable

Les variables sont présentées par leurs adresses dans le VM. Nous avons créé des fonctions `@` et `!` (fonctions de base) pour manipuler leurs valeurs :

```

1 void proc_func_at() {
2     /* read variable */
3     int iAddr = stack_pop_vm(&g_Env.StkData);

```

```

4     int ans = g_Env.VMTable.OpCodes[iAddr] ;
5     stack_push_vm(&g_Env.StkData, ans) ;
6 #ifdef DEBUG
7     printf("\n[ Debug ] A value %d is pushed into data stack\n", ans) ;
8     fflush(stdout) ;
9 #endif
10 }
```

```

1 void proc_func_exclaim() {
2     int iAddr = stack_pop_vm(&g_Env.StkData) ;
3     int iVal = stack_pop_vm(&g_Env.StkData) ;
4     g_Env.VMTable.OpCodes[iAddr] = iVal ;
5 #ifdef DEBUG
6     printf("\n[ Debug ] A value %d is set to VMTable[%d]\n", iVal, iAddr) ;
7     fflush(stdout) ;
8 #endif
9 }
```

7.3 Fonction boucle

Nous sommes intrigués par la mise en place d'une fonction boucle. Un programme en boucle se compose de trois fonctions : `while`, `loop` et `break`

1. `while` n'est pas une vraie fonction. Il est un remarque.
2. `loop` saute au dernier `while`.
3. `break` saute au prochain `loop`.

Tableau 5 présente le tableau de VM des fonctions boucle

Numéro	x	x+1	y	y+1	y+2	z	z+1
VM	WHILE	...	BREAK	z+1	...	LOOP	x
Signification	CFA		CFA			CFA	début
	de while	...	de break	fin de boucle	...	de loop	de boucle

TABLE 5 Schéma de VM pour fonction boucle

```

1 void proc_func_while() {
2     ;
3 }
4
5 void proc_func_loop() {
6 #ifdef DEBUG
7     printf("\n[ Debug ] Looping ... \n") ;
8     fflush(stdout) ;
9 #endif
10    proc_func_jr() ;
```

```

11  }
12
13 void proc_func_break() {
14 #ifdef DEBUG
15   printf("\n[ Debug ] Break ... \n");
16   fflush(stdout);
17 #endif
18   proc_func_jr();
19
20 }
```

Interprétation Ceci est inspiré par la fonctionnement de `if` et `else`. Un exemple d'exécution est présenté dans Figure 18

```

liyutong — liyutong@liyutong-Virtual-Machine: ~ — ssh -p 60001 59.78.22.1...
>>>liyutong@liyutong-Virtual-Machine:~$ Documents/CS143/cmake_build/lac
This is lac interactive mode Ver 1.1, by Yutong LI
>>>variable x

>>>5 x !

>>>essai while x @ 0 = if break else x @ 1 - x ! then x @ . loop ;

>>>essai
43210
>>>
```

FIGURE 18 Fonction boucle

7.4 Vecteur

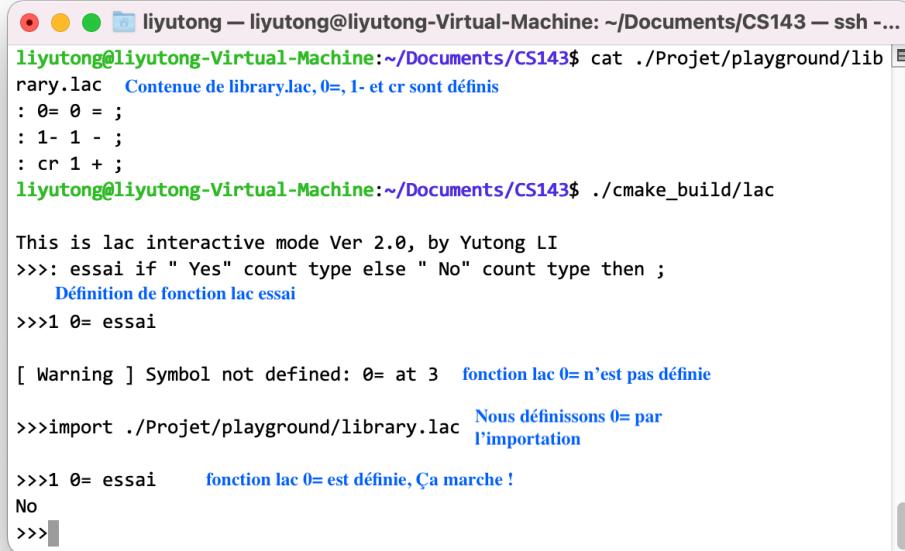
Comme pour les chaînes de caractères, `vec` saute par-dessus la zone de données dans la VM avec `jr` et place l'adresse de vecteur sur la pile de données (Tableau 6)

Numéro	x	x+1	x+2	x+3	...	x+n+3	x+n+4	x+n+5	x+n+6
VM	-2	JR	x+n+3	Vec[0]	...	0	LIT	Vec[0]	FIN
Signification	FUNC _LAC	CFA de jr	Dst de jr	Vec[0]	...	sép	CFA de (lit)	Adress de Vec	CFA de (fin)

TABLE 6 Schéma du table de VM pour un vecteur

7.5 L'importation

Comme pour python, notre interpréteur du LAC peut importer les autres fichiers par `import ${Nom de fichier}`. L'usage est indiqué dans la figure. La fonction d'importation est implémentée dans la phase d'interprétation en exécutant le fichier lac correspondant.



```

liyutong@liyutong-Virtual-Machine:~/Documents/CS143$ cat ./Projet/playground/lib
rary.lac Contenu de library.lac, 0=, 1- et cr sont définis
: 0= 0 = ;
: 1- 1 - ;
: cr 1 + ;
liyutong@liyutong-Virtual-Machine:~/Documents/CS143$ ./cmake_build/lac

This is lac interactive mode Ver 2.0, by Yutong LI
>>> essai if " Yes" count type else " No" count type then ;
    Définition de fonction lac essai
>>>1 0= essai

[ Warning ] Symbol not defined: 0= at 3  fonction lac 0= n'est pas définie
>>>import ./Projet/playground/library.lac Nous définissons 0= par
l'importation
>>>1 0= essai      fonction lac 0= est définie, Ça marche !
No
>>>

```

FIGURE 19 L'importation

7.6 Comparateur < et >

Nous définissons < et > comme deux fonctions de base :

```

1 void proc_func_gt() {
2     /* Greater than */
3     int iOp1 = stack_pop_vm(&g_Env.StkData) ;
4     int iOp2 = stack_pop_vm(&g_Env.StkData) ;
5     if (iOp1 < iOp2) {
6         stack_push_vm(&g_Env.StkData, 1) ;
7     } else {
8         stack_push_vm(&g_Env.StkData, 0) ;
9     }
10 }
11
12 void proc_func_lt() {
13     /* Less than */
14     int iOp1 = stack_pop_vm(&g_Env.StkData) ;

```

```

15     int iOp2 = stack_pop_vm(&g_Env.StkData) ;
16     if (iOp1 > iOp2) {
17         stack_push_vm(&g_Env.StkData, 1) ;
18     } else {
19         stack_push_vm(&g_Env.StkData, 0) ;
20     }
21 }
```

Ces deux fonctions sont similaires que = (Figure 20)

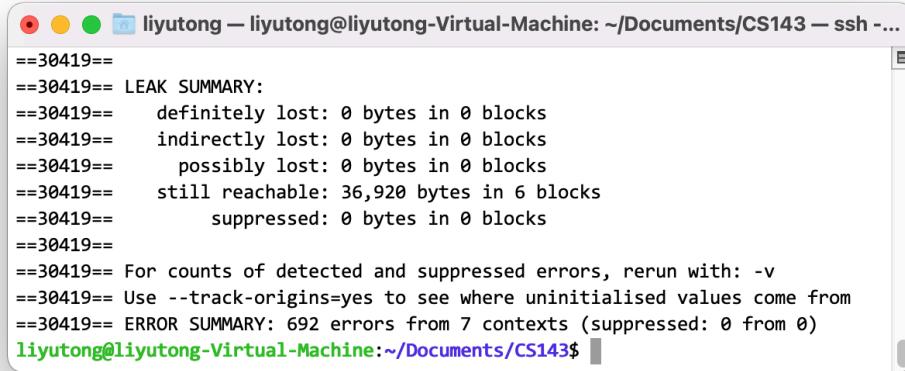
FIGURE 20 Comparateur < et > en mode interpréter

7.7 Fuite de mémoire

Nous avons testé l'interprète à l'aide de `valgrind` [1], comme le montre la Figure 21, aucune fuite de mémoire n'a été constatée.

```

1 valgrind --leak-check=full --show-reachable=yes --trace-children=yes ./build/lac ...
./Projet/playground/memchk.lac
```



A screenshot of a terminal window titled "liyutong — liyutong@liyutong-Virtual-Machine: ~/Documents/CS143 — ssh -...". The window displays the output of the valgrind command. The output shows a LEAK SUMMARY with zero bytes lost in zero blocks. It also shows an ERROR SUMMARY with 692 errors from 7 contexts, all suppressed. The command "liyutong@liyutong-Virtual-Machine:~/Documents/CS143\$" is visible at the bottom.

```
==30419==  
==30419== LEAK SUMMARY:  
==30419==   definitely lost: 0 bytes in 0 blocks  
==30419==   indirectly lost: 0 bytes in 0 blocks  
==30419==   possibly lost: 0 bytes in 0 blocks  
==30419==   still reachable: 36,920 bytes in 6 blocks  
==30419==           suppressed: 0 bytes in 0 blocks  
==30419==  
==30419== For counts of detected and suppressed errors, rerun with: -v  
==30419== Use --track-origins=yes to see where uninitialised values come from  
==30419== ERROR SUMMARY: 692 errors from 7 contexts (suppressed: 0 from 0)  
liyutong@liyutong-Virtual-Machine:~/Documents/CS143$
```

FIGURE 21 Output de valgrind, aucune fuite de mémoire

8 Jeu de LAC

Dans cette section, nous utilisons le lac pour créer un certain nombre de fonctions permettant de tester notre interpréteur

8.1 factorielle.lac

$$6! = 720$$

```

liyutong — liyutong@liyutong-Virtual-Machine: ~/Documents/CS143 — ssh -p 6000...
liyutong@liyutong-Virtual-Machine:~/Documents/CS143$ ./build/color ./Projet/playground/factorielle.lac
----- Annalex result -----
S("uu")->M(":")->N("0")->M("=")->M(";)-
->M(":-")->M("1-")->N("1")->M("-")->M(";
")->M(":-")->M("cr")->N("1")->M("+")->M(";");
->M("fact")->M("dup")->M("0=")->M("i
f")->M("drop")->N("1")->M("else")->M("dup")->M("1-")->M("recurse")->M("*")->M("then")->
M(";");
->M("go")->S("Factorielle ()->M("count")->M("type")->M("dup")->M(".")->S(") vaut :
")->M("count")->M("type")->M("fact")->M(".")->M("cr")->M(";");
->N("6")->M("go")->
-----
----- Highlight result -----
\ Fichier " factorielle.lac"
" uu"
( Ce fichier est un "exemple" étudié pour tester
l'analyseur lexical écrit en phase 1 du projet)
\ BlahBlahBlah
: 0= 0 =
: 1- 1 -
: cr 1 +
: fact ( n -- n!)
    dup 0=
    if
        drop 1
    else
        dup 1- recurse *
    then ;

: go ( n --
    " Factorielle (" count type
    dup .
    " ) vaut :
" count type
    fact . cr ;

6 go
-----
liyutong@liyutong-Virtual-Machine:~/Documents/CS143$ ./build/lac ./Projet/playground/factorielle.lac
Factorielle (6) vaut :
720
This is lac interactive mode Ver 2.0, by Yutong LI
>>>

```

FIGURE 22 Exécution de factorielle.lac

8.2 catenate.lac

"Bonjour" + "Alain!" == "BonjourAlain!"



```

---- Highlight result ----

: catenate
\ Data_Stk: [...] &input1 | &input2 |<- top
variable catenate_s1 \ char * s1
variable catenate_s2 \ char * s2
catenate_s2 ! \ s2 = & input2
catenate_s1 ! \ s1 = & input1

\ Data_Stk: [...] <- top

variable catenate_i @ 0 catenate_i ! \ int i = 0
variable catenate_j @ 0 catenate_j ! \ int j = 0

vec catenate_ans 100 \ char ans[20]

variable catenate_l1 catenate_s1 @ count catenate_l1 ! drop \ int l1 = strlen(s1)
variable catenate_l2 catenate_s2 @ count catenate_l2 ! drop \ int l2 = strlen(s2)

while
catenate_l1 @ catenate_i @ = if break then \ while (i != l1)
catenate_s1 @ catenate_i @ + @ catenate_ans catenate_j @ + ! \ ans[j] = s1[i]
catenate_i @ 1 + catenate_i ! \ i++
catenate_j @ 1 + catenate_j ! \ j++
loop

0 catenate_i ! \ i = 0

while
catenate_l2 @ catenate_i @ = if break then \ while (i != l2)
catenate_s2 @ catenate_i @ + @ catenate_ans catenate_j @ + ! \ ans[j] = s2[i]
catenate_i @ 1 + catenate_i ! \ i++
catenate_j @ 1 + catenate_j ! \ j++
loop

\ Data_Stk: [...] &input1 | &input2 |<- top
; \ return

" Bonjour" " Alain !" catenate
" The length of the first string is: " count type catenate_l1 @ . "
" count type
" The length of the second string is: " count type catenate_l2 @ . "
" count type
" The concatenated string is: " count type catenate_ans count type
-----
llyutong@llyutong-Virtual-Machine:~/Documents/CS143$ ./build/lac ./Projet/playground/catenate.lac
The length of the first string is: 7
The length of the second string is: 7
The concatenated string is: BonjourAlain !
This is lac interactive mode Ver 2.0, by Yutong LI
>>>catenate_ans catenate_ans catenate catenate_ans count type
BonjourAlain !BonjourAlain !
>>>

```

FIGURE 23 Exécution de factorielle.lac

8.3 fibonacci.lac

$$fib = [1, 1, 2, 3, 5, 8, 13, 21, \dots]$$

```

---- Highlight result ----

: fibonacci
variable fibonacci_x \ int x = input
fibonacci_x !
fibonacci_x @ 1 = if 1 else \ if (x == 1) { return 1; }
    fibonacci_x @ 2 = if 1 else \ if (x == 2) { return 1; }
        fibonacci_x @ 1 - fibonacci_x @ 2 - fibonacci swap \ else { return fib(x-1) + fib(x-2); }
        fibonacci +
    then
then
;

defer fib
' fibonacci is fib \ link function

: go_fib \ entrance
    " Fibonacci(" count type \ print information
        dup .
        " ) vaut : " count type
    fib .
;

6 go_fib
-----
liyutong@liyutong-Virtual-Machine:~/Documents/CS143$ ./build/lac ./Projet/playground/fibonacci.lac
Fibonacci(6) vaut : 8
This is lac interactive mode Ver 2.0, by Yutong LI
>>>6 fib .
8
>>>7 fib .
13
>>>8 fib .
21
>>>

```

FIGURE 24 Exécution de fibonacci.lac

8.4 sort.lac

Un algorithme Tri des bulles avec les comparateurs '<' et '>' comme fonction de base.

```

---- Highlight result ----

: sort
variable sort_vec sort_vec ! \ char * vec = input;
variable sort_len \ int len;

sort_vec @ count sort_len ! \ len = strlen(vec);
variable sort_i @ sort_i ! \ int i = 0;
variable sort_j @ sort_j ! \ int j = 0;
variable tmp @ tmp ! \ int tmp = 0;

while
sort_i @ sort_len @ = if break then \ while (i != len)
    while
        sort_j @ sort_len @ = if break then \ while (j != len)
            sort_vec @ sort_j @ + @ dup sort_vec @ sort_i @ + @ < if \ if(vec[i] < vec[j])
                sort_vec @ sort_i @ + @ \ { swap(vec[i],vec[j]); }
                sort_vec @ sort_j @ + !
                sort_vec @ sort_i @ + !
            else
                drop \ else{};
            then
                sort_j @ 1 + sort_j ! \ j++;
        loop
        sort_i @ 1 + dup sort_i ! sort_j ! \ i++ ; j = i;
    loop
; \ return ;

: go_sort \ entrance
" bubble_sort (" count type
dup count type
" ) vaut : " count type
sort sort_vec @ count type
;

" eastdfffruwieouawrbceyurug" go_sort
-----
liyutong@liyutong-Virtual-Machine:~/Documents/CS143$ ./build/lac ./Projet/playground/sort.lac
bubble_sort (eastdfffruwieouawrbceyurug) vaut : aabcdeeffffgiorrrstuuuuwy
This is lac interactive mode Ver 2.0, by Yutong LI
>>>" Bonjour, Alain!" go_sort
bubble_sort (Bonjour, Alain!) vaut : !,ABaijlnooru
>>>

```

FIGURE 25 Exécution de sort.lac

8.5 strlen.lac

Fonction de base count réalisé avec LAC

```

---- Highlight result ----

: ++ 1 + ;
: -- 1 - ;

: strlen
\ Data stack : [...] &string | <- top
variable strlen_input \ char * input;
variable strlen_cnt @ strlen_cnt ! \ int cnt = 0;

dup strlen_input ! \ input = &string;

while
strlen_input @ strlen_cnt @ + @ 0 = if break else \ while (input[cnt] != 0)
strlen_cnt @ ++ strlen_cnt ! \ cnt++;
then
loop

strlen_cnt @ \ Output to Data stack
\ Data stack : [...] &string | strlen(&string) <- top
;

: strcpy \ str2 = str1
\ Data stack : [...] str1 | str2 | <- top
variable strcpy_str1 variable strcpy_str2
strcpy_str2 ! strcpy_str1 !
variable strcpy_len1 variable strcpy_len2
variable strcpy_cnt @ strcpy_cnt !

\ Data stack : [...] <- top

strcpy_str1 @ strlen strcpy_len1 ! drop
strcpy_str2 @ strlen strcpy_len2 ! drop

\ Data stack : [...] <- top
while
strcpy_cnt @ strcpy_len1 @ = if break
else
    strcpy_str1 @ strcpy_cnt @ + @ strcpy_str2 @ strcpy_cnt @ +
    then
    strcpy_cnt @ ++ strcpy_cnt !
loop

strcpy_str1 @ strcpy_str2 @

\ Data stack : [...] str1 | str2 | <- top
;

: strcmp
\ Data stack : [...] str1 | str2 | <- top

variable strcmp_str1 variable strcmp_str2
strcmp_str2 ! strcmp_str1 !

variable strcmp_len1 variable strcmp_len2
strcmp_str1 @ strlen strcmp_len1 ! drop
strcmp_str2 @ strlen strcmp_len2 ! drop

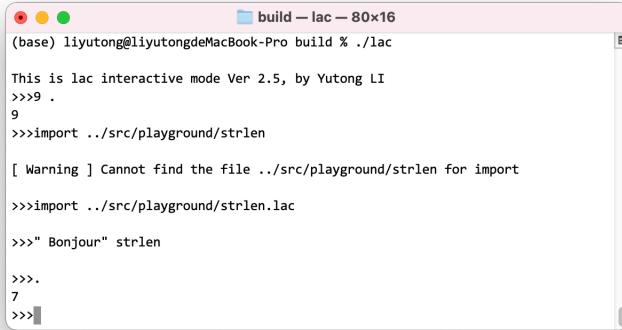
variable strcmp_cnt @ strcmp_cnt !
variable strcmp_ans 1 strcmp_ans !

strcmp_str1 @ strcmp_str2 @ = if
    while
        strcmp_str1 @ strcmp_cnt @ + strcmp_str2 @ strcmp_cnt @ + = if
        else
            0 strcmp_ans !
            break
        then
    else
        0 strcmp_ans !
    then

strcmp_str1 @ strcmp_str2 @ 0
strcmp_ans @

```

FIGURE 26 str.lac



```
(base) liyutong@liyutongdeMacBook-Pro build % ./lac
This is lac interactive mode Ver 2.5, by Yutong LI
>>> .
9
>>>import ..src/playground/strlen
[ Warning ] Cannot find the file ..src/playground/strlen for import
>>>import ..src/playground/strlen.lac
>>>" Bonjour" strlen
>>>.
7
>>>
```

FIGURE 27 Exécution de strlen

9 LAC orienté-objet

Lorsque la dernière référence à une chaîne de mots est effacée de la pile de données, la chaîne reste dans la table VM et il n'y a pas de mécanisme pour libérer cette section de mémoire. En plus, nous utilisons un algorithme de recherche linéaire pour trouver des symboles, qui pourrait probablement être amélioré en mettant en œuvre une table de hachage.

Nous voulons mettre en œuvre un interpréteur BAC plus riche, et ce travail supplémentaire se reflète dans le dossier `src-dev`. Il faut compiler ce projet de la même manière.

9.1 Modification de table de symboles

Pour rendre la recherche plus efficace, nous modifions la table de symboles pour qu'elle soit une table de hachage :

```

1  typedef struct hash_table_t {
2      int iNumEntry;
3      lac_queue_t Data[HASH_TABLE_LEN]; /* should be automatically initialized by ...
4          0 */
5  } hash_table_t;
6
7  typedef struct hash_table_entry {
8      char Key[MAX_KEY_LEN + 1];
9      void *pData;
10 } hash_table_entry;
11
12 typedef struct hash_table_query_res {
13     int idx;
14     queue_node_t *pNode;
15 } hash_table_query_res;
16
```

```

17 unsigned int hash_table_compute_hash(unsigned char *input) ;
18 hash_table_entry *hash_table_gen_entry(const char *sKey, void *pValue, int iSize) ;
20
21 bool hash_table_reset(hash_table_t *pHashTable) ;
22
23 bool hash_table_init(hash_table_t *pHashTable) ;
24
25 bool hash_table_revert(hash_table_t *pHashTable) ;
26
27 bool hash_table_checkout(hash_table_t *pHashTable) ;
28
29 bool hash_table_entry_is_empty(hash_table_t *pHashTable, int iHashTableIdx) ;
30
31 hash_table_query_res hash_table_query(hash_table_t *pHashTable, char *sKey) ;
32
33 hash_table_query_res hash_table_add(hash_table_t *pHashTable, char *sKey, void ...
    *pValue, int iSize) ;

```

Fonction `hash33` est utilisé pour la hachage. Nous pouvons aussi utiliser le MD5 algorithme avant appliquer la fonction `hash33`

9.2 Les objets

Pour permettre de libérer les chaînes de caractères qui ne sont plus utilisées, nous cherchons à remplacer le table de VM. If faut que nous créions des objets de LAC :

```

1 typedef enum e_vm_func_type {
2     /* There are three types of functions */
3     VM_FUNC_BASIC = -1,
4     VM_FUNC_LAC = -2,
5     VM_FUNC_CTRL = -3,
6 } e_vm_func_type;
7
8 typedef struct lac_object_t {
9     /* Base class */
10    int iRefCount;
11    e_lac_object_type Type;
12    char Name[MAX_LEXEME_LEN];
13    void *Child;
14 } lac_object_t;
15
16 typedef struct lac_func_t {
17     e_vm_func_type FuncType;           // only basic function has pFunc
18     basic pFunc;                    // only basic function has iCFA
19     int iCFA;                      // only basic function has iCFA
20     struct vmtable_t VMTable;
21     hash_symtable_t SymTable;
22 } lac_func_t, lac_class_t;
23
24 typedef struct lac_int_t {
25     /* lac int */
26     int iValue;
27 } lac_int_t;
28
29 typedef struct lac_var_t {
30     /* lac variable */

```

```

31     bool bValid;           /* if the variable is valid */
32     e_lac_object_type Type; /* type of variable's content */
33     lac_object_t *Object; /* pointer to the real content */
34 } lac_var_t;
35
36 typedef struct lac_vec_t {
37     /* lac vector */
38     int iRef; /* reference of this vector */
39     int iLength;
40     int iSize;
41     int *pData;
42 } lac_vec_t;

```

Interprétation : Nous avons créé une variété d'objets de LAC :

- `lac_int_t` Objets entiers, immuables
- `lac_vec_t` Objets vectoriels, modifiables
- `lac_var_t` Les objets variables, qui peuvent pointer vers un entier ou un vecteur
- `lac_func_t` Les objets de fonction, qui contiennent une fonction, qui ont leur propre table de symboles et table de VM
- `lac_class_t` les objets de type, qui peuvent générer de nouveaux objets de fonction basés sur le changement de type

L'objet vectoriel a un paramètre `iRef` qui indique la position du vecteur. Les fonctions de base ont été modifiées en conséquence, par exemple :

- Un nombre entier est ajouté à un nombre entier pour obtenir un nombre entier
- Un entier et un vecteur sont ajoutés pour modifier l'iRef du vecteur
- Le vecteur peut désormais être imprimé directement
- `fonction!` définit l'objet vers lequel la variable pointe.

9.3 LAC class

Comme le montre la figure, nous pouvons créer des instances d'un type basé sur un `class`, et les propriétés des instances et des instances sont indépendantes les unes des autres

```
(base) liytong@liytongdeMacBook-Pro build % ./lac
This is lac interactive mode Ver 3.5, by Yutong LI
>>>class point variable x variable y ;
>>>defer p ' p is point
>>>defer q ' q is point
>>>1 p.x ! 2 q.x !
[ Debug ] Variable
[ Debug ] Variable
>>>p.x @ . q.x @ .
[ Debug ] Variable
1
[ Debug ] Variable
2
>>>
```

FIGURE 28 LAC class

9.4 Gestion des objets (Garbage collection)

Comme LAC crée de nombreux objets en cours de fonctionnement, nous avons conçu un système de gestion des objets. Tous les objets nouvellement créés sont enregistrés dans une liste. Chaque objet possède un attribut `iRefCnt` pour enregistrer son compte de référence.

- Lorsque l'objet est ajouté à un table de VM ou à la pile de données, le compteur incrémenté de 1
- Lorsqu'un objet est retiré d'un table de VM ou du pile de données, le compteur diminue de 1
- Les objets ayant un compteur de 0 sont libérés(free) après chaque exécution

10 Conclusion

Avec ce projet, nous avons mis en place un interpréteur pour la langue LAC. Nous avons bien compris les mystères des langues dynamiques et compilation.

En plus des exigences établies, notre interpréteur prend en charge la mise en évidence syntaxique (code highlight), la division des entier, les variables, les déclarations de fonctions et les `while` boucles. Nous avons aussi complété une interface interactive.

Cependant, notre interpréteur LAC ne met pas en œuvre des fonctionnalités telles que les `for` boucles. Ce fonctionnalité est essentielle pour un langage de programmation complet.

Références

- [1] Valgrind™ Developers. Valgrind home. <https://www.valgrind.org/>, Nov. 2020.

11 Appendix

11.1 Exécution log du projet4.lac

```

1  :~/Documents/CS143/cmake_build$ ./projet4
2  ----- VM Table -----
3  0 :-1 1 :0 2 :-1 3 :1 4 :-1 5 :2 6 :-1 7 :3 8 :-1 9 :4 10 :-1
4
5  11 :5 12 :-1 13 :6 14 :-1 15 :7 16 :-1 17 :8 18 :-1 19 :9 20 :-1
6
7  21 :10 22 :-1 23 :11 24 :-1 25 :12 26 :-3 27 :13 28 :-3 29 :14 30 :-3
8
9  31 :15 32 :-3 33 :16 34 :-3 35 :17 36 :-3 37 :18 38 :-3 39 :19 40 :-3
10
11 41 :20 42 :-3 43 :21
12 -----
13
14 ----- Symbol Table -----
15 dup(0) -> drop(2) -> swap(4) -> .(6) -> count(8) -> type(10) -> =(12) ->
16 *(14) -> +(16) -> -(18) -> (lit)(20) -> (fin)(22) -> calculate(24) ->
17 if(26) -> else(28) -> then(30) -> recurse(32) -> @(34) -> !(36) ->
18 while(38) -> loop(40) -> break(42)
19 -----
20
21 ----- Annalex result -----
22 M(".")->M("incr")->N("1")->M("+-")->M(" ;")->M(" :")->M("2+.")->M("incr")-
23 >M("incr")->M(" .")->M(" ;")->N("7")->N("8")->M("+-")->M("2+.")->
24 -----
25
26
27 ----- Highlight result -----
28
29 : incr 1 + ;
30
31 : 2+. incr incr . ;
32
33 7 8 + 2+.
34 -----
35
36 [ Debug ] Define function incr with CFA = 44
37
38 [ Debug ] Define function 2+. with CFA = 49
39
40 [ Debug ] Push 7 into StkData
41
42 [ Debug ] Push 8 into StkData
43
44 [ Debug ] Execute basic function + at VMTable[16]
45
46 [ Debug ] Pop 8 from StkData, StkData.top = 7
47
48 [ Debug ] Pop 7 from StkData, StkData.top = N/A
49
50 [ Debug ] Push 15 into StkData
51
52 [ Debug ] Execute lac function at VMTable[49]
53
54 [ Debug ] Push 50 into StkReturn
55

```

```
56 [ Debug ] Execute lac function at VMTable[44]
57 [ Debug ] Push 45 into StkReturn
58 [ Debug ] Execute basic function (lit) at VMTable[20]
59
60 [ Debug ] Pop 45 from StkReturn, StkReturn.top = 50
61
62 [ Debug ] Push 46 into StkReturn
63
64 [ Debug ] Push 1 into StkData
65
66 [ Debug ] Pop 46 from StkReturn, StkReturn.top = 50
67
68 [ Debug ] Push 47 into StkReturn
69
70 [ Debug ] Execute VMTable[47]
71
72 [ Debug ] Execute basic function + at VMTable[16]
73
74 [ Debug ] Pop 1 from StkData, StkData.top = 15
75
76 [ Debug ] Push 16 into StkData
77
78 [ Debug ] Pop 15 from StkData, StkData.top = N/A
79
80 [ Debug ] Push 48 into StkReturn
81
82 [ Debug ] Execute VMTable[48]
83
84 [ Debug ] Execute basic function (fin) at VMTable[22]
85
86 [ Debug ] Pop 48 from StkReturn, StkReturn.top = 50
87
88 [ Debug ] Push 50 into StkReturn
89
90 [ Debug ] Pop 50 from StkReturn, StkReturn.top = N/A
91
92 [ Debug ] Push 51 into StkReturn
93
94 [ Debug ] Execute VMTable[51]
95
96 [ Debug ] Execute lac function at VMTable[44]
97
98 [ Debug ] Push 45 into StkReturn
99
100 [ Debug ] Execute basic function (lit) at VMTable[20]
101
102 [ Debug ] Pop 45 from StkReturn, StkReturn.top = 51
103
104 [ Debug ] Push 46 into StkReturn
105
106 [ Debug ] Push 1 into StkData
107
108 [ Debug ] Pop 46 from StkReturn, StkReturn.top = 51
109
110 [ Debug ] Push 47 into StkReturn
111
112 [ Debug ] Execute VMTable[47]
113
114 [ Debug ] Execute basic function + at VMTable[16]
115
116 [ Debug ] Push 48 into StkReturn
117
```

```

118 [ Debug ] Pop 1 from StkData, StkData.top = 16
119 [ Debug ] Pop 16 from StkData, StkData.top = N/A
120 [ Debug ] Push 17 into StkData
121
122 [ Debug ] Pop 47 from StkReturn, StkReturn.top = 51
123
124 [ Debug ] Push 48 into StkReturn
125
126 [ Debug ] Execute VMTable[48]
127
128 [ Debug ] Execute basic function (fin) at VMTable[22]
129
130 [ Debug ] Execute basic function . at VMTable[6]
131
132 [ Debug ] Pop 48 from StkReturn, StkReturn.top = 51
133
134 [ Debug ] Pop 51 from StkReturn, StkReturn.top = N/A
135
136 [ Debug ] Push 52 into StkReturn
137
138 [ Debug ] Execute VMTable[52]
139
140 [ Debug ] Execute basic function . at VMTable[6]
141
142 [ Debug ] Pop 17 from StkData, StkData.top = N/A
143 17
144 [ Debug ] Pop 52 from StkReturn, StkReturn.top = N/A
145
146 [ Debug ] Push 53 into StkReturn
147
148 [ Debug ] Execute VMTable[53]
149
150 [ Debug ] Execute basic function (fin) at VMTable[22]
151
152 [ Debug ] Pop 53 from StkReturn, StkReturn.top = N/A
153
154 [ Info ] Execution finished

```

11.2 Exécution log du factorielle.lac

```

1 liyutong@liyutong - Virtual - Machine :~/Documents/CS143$ ./build/lac ...
2 ./Projet/playground/factorielle.lac
3 -----
4 ----- VM Table -----
5 0 :-1 1 :0 2 :-1 3 :1 4 :-1 5 :2 6 :-1 7 :3 8 :-1 9 :4 10 :-1
6
7 11 :5 12 :-1 13 :6 14 :-1 15 :7 16 :-1 17 :8 18 :-1 19 :9 20 :-1
8
9 21 :10 22 :-1 23 :11 24 :-1 25 :12 26 :-1 27 :13 28 :-1 29 :14 30 :-1
10
11 31 :15 32 :-3 33 :16 34 :-3 35 :17 36 :-3 37 :18 38 :-3 39 :19 40 :-3
12
13 41 :20 42 :-3 43 :21 44 :-3 45 :22 46 :-3 47 :23 48 :-3 49 :24 50 :-3
14 -----
15 ----- Symbol Table -----
16

```

```

17 dup(0) -> drop(2) -> swap(4) -> .(6) -> count(8) -> type(10) -> =(12) -> >(14) ...
    -> <(16) -> *(18) -> +(20) -> -(22) -> /(24) -> (lit)(26) -> (fin)(28) -> ...
        calculate(30) -> if(32) -> else(34) -> then(36) -> recurse(38) -> @!(40) -> ...
            !(42) -> while(44) -> loop(46) -> break(48) -> jr(50)
18 -----
19
20 ---- Annalex result ----
21 S("uu")->M(" :")->M("0=")->N("0")->M(" =")->M(" ;")->M("1-")
22 ->N("1")->M(" -")->M(" ;")->M(" cr")->N("1")->M(" +")->M(" ;")
23 ->M(" :")->M(" fact")->M("dup")->M("0=")->M(" if")->M("drop")->N("1")
24 ->M(" else")->M("dup")->M("1-")->M(" recurse")->M(" *")->M(" then")->
25 M(" ;")->M(" :")->M(" go")->S(" Factorielle ...
    ") ->M("count")->M("type")->M("dup")->M(" .")->S(" vaut :
26 ") ->M("count")->M("type")->M(" fact")->M(" .")->M(" cr")->M(" ;")->N("6")->M("go")->
27 -----
28
29
30 ---- Highlight result ----
31
32 \ Fichier " factorielle.lac"
33 " uu"
34 ( Ce fichier est un "exemple" étudié pour tester
35 l'analyseur lexical écrit en phase 1 du projet)
36 \ BlahBlahBlah
37 : 0= 0 = ;
38 : 1- 1 - ;
39 : cr 1 + ;
40 : fact ( n -- n !)
41     dup 0=
42     if
43         drop 1
44     else
45         dup 1- recurse *
46     then ;
47
48 : go ( n -- )
49     " Factorielle (" count type
50     dup .
51     " ) vaut :
52 " count type
53     fact . cr ;
54
55 6 go
56 -----
57
58 [ Info ] 46 lexemes to process
59
60 [ Info ] A string uu is added to VMTable [52]
61
62 [ Debug ] Push 52 into StkData
63
64 [ Debug ] Define function 0= with CFA = 55
65
66 [ Debug ] Define function 1- with CFA = 60
67
68 [ Debug ] Define function cr with CFA = 65
69
70 [ Debug ] Define function fact with CFA = 70
71
72 [ Debug ] Define function go with CFA = 85
73
74 [ Debug ] Push 6 into StkData

```

```
75 [ Debug ] Execute lac function at VMTable[85]
76 [ Debug ] Push 86 into StkReturn
77 [ Debug ] Previous lac function execute VMTable[86] , VMTable[86] = 50
78 [ Debug ] Execute basic function jr at VMTable[50]
79 [ Debug ] Pop 86 from StkReturn , StkReturn.top = N/A
80 [ Debug ] Push 101 into StkReturn
81 [ Debug ] Unconditional jump to 101 from 86
82 [ Debug ] Pop 101 from StkReturn , StkReturn.top = N/A
83 [ Debug ] Push 102 into StkReturn
84 [ Debug ] Previous lac function execute VMTable[102] , VMTable[102] = 26
85 [ Debug ] Execute basic function (lit) at VMTable[26]
86 [ Debug ] Pop 102 from StkReturn , StkReturn.top = N/A
87 [ Debug ] Push 103 into StkReturn
88 [ Debug ] Push 88 into StkData
89 [ Debug ] Fonction (lit) executed , 88 is read to StkData
90 [ Debug ] Pop 103 from StkReturn , StkReturn.top = N/A
91 [ Debug ] Push 104 into StkReturn
92 [ Debug ] Previous lac function execute VMTable[104] , VMTable[104] = 8
93 [ Debug ] Execute basic function count at VMTable[8]
94 [ Debug ] Push 13 into StkData
95 [ Debug ] Pop 104 from StkReturn , StkReturn.top = N/A
96 [ Debug ] Push 105 into StkReturn
97 [ Debug ] Previous lac function execute VMTable[105] , VMTable[105] = 10
98 [ Debug ] Execute basic function type at VMTable[10]
99 [ Debug ] Pop 13 from StkData , StkData.top = 88
100 [ Debug ] Pop 88 from StkData , StkData.top = 6
101 Factorielle (
102 [ Debug ] Pop 105 from StkReturn , StkReturn.top = N/A
103 [ Debug ] Push 106 into StkReturn
104 [ Debug ] Previous lac function execute VMTable[106] , VMTable[106] = 0
105 [ Debug ] Execute basic function dup at VMTable[0]
106 [ Debug ] Push 6 into StkData
```

```
137 [ Debug ] Pop 106 from StkReturn , StkReturn.top = N/A
138 [ Debug ] Push 107 into StkReturn
139
140 [ Debug ] Previous lac function execute VMTable[107] , VMTable[107] = 6
141
142 [ Debug ] Execute basic function . at VMTable[6]
143
144 [ Debug ] Pop 6 from StkData , StkData.top = 6
145
146 [ Debug ] Push 107 into StkReturn , StkReturn.top = N/A
147 6
148 [ Debug ] Pop 107 from StkReturn , StkReturn.top = N/A
149
150 [ Debug ] Push 108 into StkReturn
151
152 [ Debug ] Previous lac function execute VMTable[108] , VMTable[108] = 50
153
154 [ Debug ] Execute basic function jr at VMTable[50]
155
156 [ Debug ] Pop 108 from StkReturn , StkReturn.top = N/A
157
158 [ Debug ] Push 120 into StkReturn
159
160 [ Debug ] Unconditional jump to 120 from 108
161
162 [ Debug ] Pop 120 from StkReturn , StkReturn.top = N/A
163
164 [ Debug ] Push 121 into StkReturn
165
166 [ Debug ] Previous lac function execute VMTable[121] , VMTable[121] = 26
167
168 [ Debug ] Execute basic function (lit) at VMTable[26]
169
170 [ Debug ] Pop 121 from StkReturn , StkReturn.top = N/A
171
172 [ Debug ] Push 122 into StkReturn
173
174 [ Debug ] Push 110 into StkData
175
176 [ Debug ] Fonction (lit) executed , 110 is read to StkData
177
178 [ Debug ] Pop 122 from StkReturn , StkReturn.top = N/A
179
180 [ Debug ] Push 123 into StkReturn
181
182 [ Debug ] Previous lac function execute VMTable[123] , VMTable[123] = 8
183
184 [ Debug ] Execute basic function count at VMTable[8]
185
186 [ Debug ] Push 10 into StkData
187
188 [ Debug ] Pop 123 from StkReturn , StkReturn.top = N/A
189
190 [ Debug ] Push 124 into StkReturn
191
192 [ Debug ] Previous lac function execute VMTable[124] , VMTable[124] = 10
193
194 [ Debug ] Execute basic function type at VMTable[10]
195
196 [ Debug ] Pop 10 from StkData , StkData.top = 110
197
198 [ Debug ] Pop 110 from StkData , StkData.top = 6
```

```

199  ) vaut :
200 [ Debug ] Pop 124 from StkReturn , StkReturn.top = N/A
202
203 [ Debug ] Push 125 into StkReturn
204
205 [ Debug ] Previous lac function execute VMTable[125] , VMTable[125] = 70
206
207 [ Debug ] Execute lac function at VMTable[70]
208
209 [ Debug ] Push 71 into StkReturn
210
211 [ Debug ] Previous lac function execute VMTable[71] , VMTable[71] = 0
212
213 [ Debug ] Execute basic function dup at VMTable[0]
214
215 [ Debug ] Push 6 into StkData
216
217 [ Debug ] Pop 71 from StkReturn , StkReturn.top = 125
218
219 [ Debug ] Push 72 into StkReturn
220
221 [ Debug ] Previous lac function execute VMTable[72] , VMTable[72] = 55
222
223 [ Debug ] Execute lac function at VMTable[55]
224
225 [ Debug ] Push 56 into StkReturn
226
227 [ Debug ] Previous lac function execute VMTable[56] , VMTable[56] = 26
228
229 [ Debug ] Execute basic function ( lit ) at VMTable[26]
230
231 [ Debug ] Pop 56 from StkReturn , StkReturn.top = 72
232
233 [ Debug ] Push 57 into StkReturn
234
235 [ Debug ] Push 0 into StkData
236
237 [ Debug ] Fonction ( lit ) executed , 0 is read to StkData
238
239 [ Debug ] Pop 57 from StkReturn , StkReturn.top = 72
240
241 [ Debug ] Push 58 into StkReturn
242
243 [ Debug ] Previous lac function execute VMTable[58] , VMTable[58] = 12
244
245 [ Debug ] Execute basic function = at VMTable[12]
246
247 [ Debug ] Pop 0 from StkData , StkData.top = 6
248
249 [ Debug ] Pop 6 from StkData , StkData.top = 6
250
251 [ Debug ] Push 0 into StkData
252
253 [ Debug ] Pop 58 from StkReturn , StkReturn.top = 72
254
255 [ Debug ] Push 59 into StkReturn
256
257 [ Debug ] Previous lac function execute VMTable[59] , VMTable[59] = 28
258
259 [ Debug ] Execute basic function ( fin ) at VMTable[28]
260

```

```
261 [ Debug ] Pop 59 from StkReturn, StkReturn.top = 72
262 [ Debug ] Fonction (fin) executed, 59 is erased from StkReturn
264
265 [ Debug ] Pop 72 from StkReturn, StkReturn.top = 125
266
267 [ Debug ] Push 73 into StkReturn
268
269 [ Debug ] Previous lac function execute VMTable[73], VMTable[73] = 32
270
271 [ Debug ] Execute basic function if at VMTable[32]
272
273 [ Debug ] Pop 0 from StkData, StkData.top = 6
274
275 [ Debug ] Pop 73 from StkReturn, StkReturn.top = 125
276
277 [ Debug ] Push 79 into StkReturn
278
279 [ Debug ] Function If is not taken, jumps to 79 from 73
280
281 [ Debug ] Pop 79 from StkReturn, StkReturn.top = 125
282
283 [ Debug ] Push 80 into StkReturn
284
285 [ Debug ] Previous lac function execute VMTable[80], VMTable[80] = 0
286
287 [ Debug ] Execute basic function dup at VMTable[0]
288
289 [ Debug ] Push 6 into StkData
290
291 [ Debug ] Pop 80 from StkReturn, StkReturn.top = 125
292
293 [ Debug ] Push 81 into StkReturn
294
295 [ Debug ] Previous lac function execute VMTable[81], VMTable[81] = 60
296
297 [ Debug ] Execute lac function at VMTable[60]
298
299 [ Debug ] Push 61 into StkReturn
300
301 [ Debug ] Previous lac function execute VMTable[61], VMTable[61] = 26
302
303 [ Debug ] Execute basic function (lit) at VMTable[26]
304
305 [ Debug ] Pop 61 from StkReturn, StkReturn.top = 81
306
307 [ Debug ] Push 62 into StkReturn
308
309 [ Debug ] Push 1 into StkData
310
311 [ Debug ] Fonction (lit) executed, 1 is read to StkData
312
313 [ Debug ] Pop 62 from StkReturn, StkReturn.top = 81
314
315 [ Debug ] Push 63 into StkReturn
316
317 [ Debug ] Previous lac function execute VMTable[63], VMTable[63] = 22
318
319 [ Debug ] Execute basic function - at VMTable[22]
320
321 [ Debug ] Pop 1 from StkData, StkData.top = 6
322
```

```

323 [ Debug ] Pop 6 from StkData, StkData.top = 6
324 [ Debug ] Push 5 into StkData
326
327 [ Debug ] Pop 63 from StkReturn, StkReturn.top = 81
328
329 [ Debug ] Push 64 into StkReturn
330
331 [ Debug ] Previous lac function execute VMTable[64], VMTable[64] = 28
332
333 [ Debug ] Execute basic function (fin) at VMTable[28]
334
335 [ Debug ] Pop 64 from StkReturn, StkReturn.top = 81
336
337 [ Debug ] Fonction (fin) executed, 64 is erased from StkReturn
338
339 [ Debug ] Pop 81 from StkReturn, StkReturn.top = 125
340
341 [ Debug ] Push 82 into StkReturn
342
343 [ Debug ] Previous lac function execute VMTable[82], VMTable[82] = 70
344
345 [ Debug ] Execute lac function at VMTable[70]
346
347 [ Debug ] Push 71 into StkReturn
348
349 [ Debug ] Previous lac function execute VMTable[71], VMTable[71] = 0
350
351 [ Debug ] Execute basic function dup at VMTable[0]
352
353 [ Debug ] Push 5 into StkData
354
355 [ Debug ] Pop 71 from StkReturn, StkReturn.top = 82
356
357 [ Debug ] Push 72 into StkReturn
358
359 [ Debug ] Previous lac function execute VMTable[72], VMTable[72] = 55
360
361 [ Debug ] Execute lac function at VMTable[55]
362
363 [ Debug ] Push 56 into StkReturn
364
365 [ Debug ] Previous lac function execute VMTable[56], VMTable[56] = 26
366
367 [ Debug ] Execute basic function (lit) at VMTable[26]
368
369 [ Debug ] Pop 56 from StkReturn, StkReturn.top = 72
370
371 [ Debug ] Push 57 into StkReturn
372
373 [ Debug ] Push 0 into StkData
374
375 [ Debug ] Fonction (lit) executed, 0 is read to StkData
376
377 [ Debug ] Pop 57 from StkReturn, StkReturn.top = 72
378
379 [ Debug ] Push 58 into StkReturn
380
381 [ Debug ] Previous lac function execute VMTable[58], VMTable[58] = 12
382
383 [ Debug ] Execute basic function = at VMTable[12]
384

```

```
385 [ Debug ] Pop 0 from StkData, StkData.top = 5
386
387 [ Debug ] Pop 5 from StkData, StkData.top = 5
388
389 [ Debug ] Push 0 into StkData
390
391 [ Debug ] Pop 58 from StkReturn, StkReturn.top = 72
392
393 [ Debug ] Push 59 into StkReturn
394
395 [ Debug ] Previous lac function execute VMTable[59], VMTable[59] = 28
396
397 [ Debug ] Execute basic function (fin) at VMTable[28]
398
399 [ Debug ] Pop 59 from StkReturn, StkReturn.top = 72
400
401 [ Debug ] Fonction (fin) executed, 59 is erased from StkReturn
402
403 [ Debug ] Pop 72 from StkReturn, StkReturn.top = 82
404
405 [ Debug ] Push 73 into StkReturn
406
407 [ Debug ] Previous lac function execute VMTable[73], VMTable[73] = 32
408
409 [ Debug ] Execute basic function if at VMTable[32]
410
411 [ Debug ] Pop 0 from StkData, StkData.top = 5
412
413 [ Debug ] Pop 73 from StkReturn, StkReturn.top = 82
414
415 [ Debug ] Push 79 into StkReturn
416
417 [ Debug ] Function If is not taken, jumps to 79 from 73
418
419 [ Debug ] Pop 79 from StkReturn, StkReturn.top = 82
420
421 [ Debug ] Push 80 into StkReturn
422
423 [ Debug ] Previous lac function execute VMTable[80], VMTable[80] = 0
424
425 [ Debug ] Execute basic function dup at VMTable[0]
426
427 [ Debug ] Push 5 into StkData
428
429 [ Debug ] Pop 80 from StkReturn, StkReturn.top = 82
430
431 [ Debug ] Push 81 into StkReturn
432
433 [ Debug ] Previous lac function execute VMTable[81], VMTable[81] = 60
434
435 [ Debug ] Execute lac function at VMTable[60]
436
437 [ Debug ] Push 61 into StkReturn
438
439 [ Debug ] Previous lac function execute VMTable[61], VMTable[61] = 26
440
441 [ Debug ] Execute basic function (lit) at VMTable[26]
442
443 [ Debug ] Pop 61 from StkReturn, StkReturn.top = 81
444
445 [ Debug ] Push 62 into StkReturn
446
```

```
447 [ Debug ] Push 1 into StkData
448 [ Debug ] Fonction (lit) executed , 1 is read to StkData
450
451 [ Debug ] Pop 62 from StkReturn , StkReturn.top = 81
452
453 [ Debug ] Push 63 into StkReturn
454
455 [ Debug ] Previous lac function execute VMTable[63] , VMTable[63] = 22
456
457 [ Debug ] Execute basic function - at VMTable[22]
458
459 [ Debug ] Pop 1 from StkData , StkData.top = 5
460
461 [ Debug ] Pop 5 from StkData , StkData.top = 5
462
463 [ Debug ] Push 4 into StkData
464
465 [ Debug ] Pop 63 from StkReturn , StkReturn.top = 81
466
467 [ Debug ] Push 64 into StkReturn
468
469 [ Debug ] Previous lac function execute VMTable[64] , VMTable[64] = 28
470
471 [ Debug ] Execute basic function (fin) at VMTable[28]
472
473 [ Debug ] Pop 64 from StkReturn , StkReturn.top = 81
474
475 [ Debug ] Fonction (fin) executed , 64 is erased from StkReturn
476
477 [ Debug ] Pop 81 from StkReturn , StkReturn.top = 82
478
479 [ Debug ] Push 82 into StkReturn
480
481 [ Debug ] Previous lac function execute VMTable[82] , VMTable[82] = 70
482
483 [ Debug ] Execute lac function at VMTable[70]
484
485 [ Debug ] Push 71 into StkReturn
486
487 [ Debug ] Previous lac function execute VMTable[71] , VMTable[71] = 0
488
489 [ Debug ] Execute basic function dup at VMTable[0]
490
491 [ Debug ] Push 4 into StkData
492
493 [ Debug ] Pop 71 from StkReturn , StkReturn.top = 82
494
495 [ Debug ] Push 72 into StkReturn
496
497 [ Debug ] Previous lac function execute VMTable[72] , VMTable[72] = 55
498
499 [ Debug ] Execute lac function at VMTable[55]
500
501 [ Debug ] Push 56 into StkReturn
502
503 [ Debug ] Previous lac function execute VMTable[56] , VMTable[56] = 26
504
505 [ Debug ] Execute basic function (lit) at VMTable[26]
506
507 [ Debug ] Pop 56 from StkReturn , StkReturn.top = 72
508
```

```
509 [ Debug ] Push 57 into StkReturn
510 [ Debug ] Push 0 into StkData
512
513 [ Debug ] Fonction (lit) executed, 0 is read to StkData
514
515 [ Debug ] Pop 57 from StkReturn, StkReturn.top = 72
516
517 [ Debug ] Push 58 into StkReturn
518
519 [ Debug ] Previous lac function execute VMTable[58], VMTable[58] = 12
520
521 [ Debug ] Execute basic function = at VMTable[12]
522
523 [ Debug ] Pop 0 from StkData, StkData.top = 4
524
525 [ Debug ] Pop 4 from StkData, StkData.top = 4
526
527 [ Debug ] Push 0 into StkData
528
529 [ Debug ] Pop 58 from StkReturn, StkReturn.top = 72
530
531 [ Debug ] Push 59 into StkReturn
532
533 [ Debug ] Previous lac function execute VMTable[59], VMTable[59] = 28
534
535 [ Debug ] Execute basic function (fin) at VMTable[28]
536
537 [ Debug ] Pop 59 from StkReturn, StkReturn.top = 72
538
539 [ Debug ] Fonction (fin) executed, 59 is erased from StkReturn
540
541 [ Debug ] Pop 72 from StkReturn, StkReturn.top = 82
542
543 [ Debug ] Push 73 into StkReturn
544
545 [ Debug ] Previous lac function execute VMTable[73], VMTable[73] = 32
546
547 [ Debug ] Execute basic function if at VMTable[32]
548
549 [ Debug ] Pop 0 from StkData, StkData.top = 4
550
551 [ Debug ] Pop 73 from StkReturn, StkReturn.top = 82
552
553 [ Debug ] Push 79 into StkReturn
554
555 [ Debug ] Function If is not taken, jumps to 79 from 73
556
557 [ Debug ] Pop 79 from StkReturn, StkReturn.top = 82
558
559 [ Debug ] Push 80 into StkReturn
560
561 [ Debug ] Previous lac function execute VMTable[80], VMTable[80] = 0
562
563 [ Debug ] Execute basic function dup at VMTable[0]
564
565 [ Debug ] Push 4 into StkData
566
567 [ Debug ] Pop 80 from StkReturn, StkReturn.top = 82
568
569 [ Debug ] Push 81 into StkReturn
570
```

```
571 [ Debug ] Previous lac function execute VMTable[81], VMTable[81] = 60
572 [ Debug ] Execute lac function at VMTable[60]
574
575 [ Debug ] Push 61 into StkReturn
576
577 [ Debug ] Previous lac function execute VMTable[61], VMTable[61] = 26
578
579 [ Debug ] Execute basic function (lit) at VMTable[26]
580
581 [ Debug ] Pop 61 from StkReturn, StkReturn.top = 81
582
583 [ Debug ] Push 62 into StkReturn
584
585 [ Debug ] Push 1 into StkData
586
587 [ Debug ] Fonction (lit) executed, 1 is read to StkData
588
589 [ Debug ] Pop 62 from StkReturn, StkReturn.top = 81
590
591 [ Debug ] Push 63 into StkReturn
592
593 [ Debug ] Previous lac function execute VMTable[63], VMTable[63] = 22
594
595 [ Debug ] Execute basic function - at VMTable[22]
596
597 [ Debug ] Pop 1 from StkData, StkData.top = 4
598
599 [ Debug ] Pop 4 from StkData, StkData.top = 4
600
601 [ Debug ] Push 3 into StkData
602
603 [ Debug ] Pop 63 from StkReturn, StkReturn.top = 81
604
605 [ Debug ] Push 64 into StkReturn
606
607 [ Debug ] Previous lac function execute VMTable[64], VMTable[64] = 28
608
609 [ Debug ] Execute basic function (fin) at VMTable[28]
610
611 [ Debug ] Pop 64 from StkReturn, StkReturn.top = 81
612
613 [ Debug ] Fonction (fin) executed, 64 is erased from StkReturn
614
615 [ Debug ] Pop 81 from StkReturn, StkReturn.top = 82
616
617 [ Debug ] Push 82 into StkReturn
618
619 [ Debug ] Previous lac function execute VMTable[82], VMTable[82] = 70
620
621 [ Debug ] Execute lac function at VMTable[70]
622
623 [ Debug ] Push 71 into StkReturn
624
625 [ Debug ] Previous lac function execute VMTable[71], VMTable[71] = 0
626
627 [ Debug ] Execute basic function dup at VMTable[0]
628
629 [ Debug ] Push 3 into StkData
630
631 [ Debug ] Pop 71 from StkReturn, StkReturn.top = 82
632
```

```
633 [ Debug ] Push 72 into StkReturn
634 [ Debug ] Previous lac function execute VMTable[72] , VMTable[72] = 55
636
637 [ Debug ] Execute lac function at VMTable[55]
638
639 [ Debug ] Push 56 into StkReturn
640
641 [ Debug ] Previous lac function execute VMTable[56] , VMTable[56] = 26
642
643 [ Debug ] Execute basic function (lit) at VMTable[26]
644
645 [ Debug ] Pop 56 from StkReturn , StkReturn.top = 72
646
647 [ Debug ] Push 57 into StkReturn
648
649 [ Debug ] Push 0 into StkData
650
651 [ Debug ] Fonction (lit) executed , 0 is read to StkData
652
653 [ Debug ] Pop 57 from StkReturn , StkReturn.top = 72
654
655 [ Debug ] Push 58 into StkReturn
656
657 [ Debug ] Previous lac function execute VMTable[58] , VMTable[58] = 12
658
659 [ Debug ] Execute basic function = at VMTable[12]
660
661 [ Debug ] Pop 0 from StkData , StkData.top = 3
662
663 [ Debug ] Pop 3 from StkData , StkData.top = 3
664
665 [ Debug ] Push 0 into StkData
666
667 [ Debug ] Pop 58 from StkReturn , StkReturn.top = 72
668
669 [ Debug ] Push 59 into StkReturn
670
671 [ Debug ] Previous lac function execute VMTable[59] , VMTable[59] = 28
672
673 [ Debug ] Execute basic function (fin) at VMTable[28]
674
675 [ Debug ] Pop 59 from StkReturn , StkReturn.top = 72
676
677 [ Debug ] Fonction (fin) executed , 59 is erased from StkReturn
678
679 [ Debug ] Pop 72 from StkReturn , StkReturn.top = 82
680
681 [ Debug ] Push 73 into StkReturn
682
683 [ Debug ] Previous lac function execute VMTable[73] , VMTable[73] = 32
684
685 [ Debug ] Execute basic function if at VMTable[32]
686
687 [ Debug ] Pop 0 from StkData , StkData.top = 3
688
689 [ Debug ] Pop 73 from StkReturn , StkReturn.top = 82
690
691 [ Debug ] Push 79 into StkReturn
692
693 [ Debug ] Function If is not taken , jumps to 79 from 73
694
```

```
695 [ Debug ] Pop 79 from StkReturn , StkReturn.top = 82
696 [ Debug ] Push 80 into StkReturn
698
699 [ Debug ] Previous lac function execute VMTable[80] , VMTable[80] = 0
700
701 [ Debug ] Execute basic function dup at VMTable[0]
702
703 [ Debug ] Push 3 into StkData
704
705 [ Debug ] Pop 80 from StkReturn , StkReturn.top = 82
706
707 [ Debug ] Push 81 into StkReturn
708
709 [ Debug ] Previous lac function execute VMTable[81] , VMTable[81] = 60
710
711 [ Debug ] Execute lac function at VMTable[60]
712
713 [ Debug ] Push 61 into StkReturn
714
715 [ Debug ] Previous lac function execute VMTable[61] , VMTable[61] = 26
716
717 [ Debug ] Execute basic function (lit) at VMTable[26]
718
719 [ Debug ] Pop 61 from StkReturn , StkReturn.top = 81
720
721 [ Debug ] Push 62 into StkReturn
722
723 [ Debug ] Push 1 into StkData
724
725 [ Debug ] Fonction (lit) executed , 1 is read to StkData
726
727 [ Debug ] Pop 62 from StkReturn , StkReturn.top = 81
728
729 [ Debug ] Push 63 into StkReturn
730
731 [ Debug ] Previous lac function execute VMTable[63] , VMTable[63] = 22
732
733 [ Debug ] Execute basic function - at VMTable[22]
734
735 [ Debug ] Pop 1 from StkData , StkData.top = 3
736
737 [ Debug ] Pop 3 from StkData , StkData.top = 3
738
739 [ Debug ] Push 2 into StkData
740
741 [ Debug ] Pop 63 from StkReturn , StkReturn.top = 81
742
743 [ Debug ] Push 64 into StkReturn
744
745 [ Debug ] Previous lac function execute VMTable[64] , VMTable[64] = 28
746
747 [ Debug ] Execute basic function (fin) at VMTable[28]
748
749 [ Debug ] Pop 64 from StkReturn , StkReturn.top = 81
750
751 [ Debug ] Fonction (fin) executed , 64 is erased from StkReturn
752
753 [ Debug ] Pop 81 from StkReturn , StkReturn.top = 82
754
755 [ Debug ] Push 82 into StkReturn
756
```

```
757 [ Debug ] Previous lac function execute VMTable[82], VMTable[82] = 70
758 [ Debug ] Execute lac function at VMTable[70]
760
761 [ Debug ] Push 71 into StkReturn
762
763 [ Debug ] Previous lac function execute VMTable[71], VMTable[71] = 0
764
765 [ Debug ] Execute basic function dup at VMTable[0]
766
767 [ Debug ] Push 2 into StkData
768
769 [ Debug ] Pop 71 from StkReturn, StkReturn.top = 82
770
771 [ Debug ] Push 72 into StkReturn
772
773 [ Debug ] Previous lac function execute VMTable[72], VMTable[72] = 55
774
775 [ Debug ] Execute lac function at VMTable[55]
776
777 [ Debug ] Push 56 into StkReturn
778
779 [ Debug ] Previous lac function execute VMTable[56], VMTable[56] = 26
780
781 [ Debug ] Execute basic function (lit) at VMTable[26]
782
783 [ Debug ] Pop 56 from StkReturn, StkReturn.top = 72
784
785 [ Debug ] Push 57 into StkReturn
786
787 [ Debug ] Push 0 into StkData
788
789 [ Debug ] Fonction (lit) executed, 0 is read to StkData
790
791 [ Debug ] Pop 57 from StkReturn, StkReturn.top = 72
792
793 [ Debug ] Push 58 into StkReturn
794
795 [ Debug ] Previous lac function execute VMTable[58], VMTable[58] = 12
796
797 [ Debug ] Execute basic function = at VMTable[12]
798
799 [ Debug ] Pop 0 from StkData, StkData.top = 2
800
801 [ Debug ] Pop 2 from StkData, StkData.top = 2
802
803 [ Debug ] Push 0 into StkData
804
805 [ Debug ] Pop 58 from StkReturn, StkReturn.top = 72
806
807 [ Debug ] Push 59 into StkReturn
808
809 [ Debug ] Previous lac function execute VMTable[59], VMTable[59] = 28
810
811 [ Debug ] Execute basic function (fin) at VMTable[28]
812
813 [ Debug ] Pop 59 from StkReturn, StkReturn.top = 72
814
815 [ Debug ] Fonction (fin) executed, 59 is erased from StkReturn
816
817 [ Debug ] Pop 72 from StkReturn, StkReturn.top = 82
818
```

```
819 [ Debug ] Push 73 into StkReturn
820 [ Debug ] Previous lac function execute VMTable[73] , VMTable[73] = 32
822
823 [ Debug ] Execute basic function if at VMTable[32]
824
825 [ Debug ] Pop 0 from StkData , StkData.top = 2
826
827 [ Debug ] Pop 73 from StkReturn , StkReturn.top = 82
828
829 [ Debug ] Push 79 into StkReturn
830
831 [ Debug ] Function If is not taken , jumps to 79 from 73
832
833 [ Debug ] Pop 79 from StkReturn , StkReturn.top = 82
834
835 [ Debug ] Push 80 into StkReturn
836
837 [ Debug ] Previous lac function execute VMTable[80] , VMTable[80] = 0
838
839 [ Debug ] Execute basic function dup at VMTable[0]
840
841 [ Debug ] Push 2 into StkData
842
843 [ Debug ] Pop 80 from StkReturn , StkReturn.top = 82
844
845 [ Debug ] Push 81 into StkReturn
846
847 [ Debug ] Previous lac function execute VMTable[81] , VMTable[81] = 60
848
849 [ Debug ] Execute lac function at VMTable[60]
850
851 [ Debug ] Push 61 into StkReturn
852
853 [ Debug ] Previous lac function execute VMTable[61] , VMTable[61] = 26
854
855 [ Debug ] Execute basic function (lit) at VMTable[26]
856
857 [ Debug ] Pop 61 from StkReturn , StkReturn.top = 81
858
859 [ Debug ] Push 62 into StkReturn
860
861 [ Debug ] Push 1 into StkData
862
863 [ Debug ] Fonction (lit) executed , 1 is read to StkData
864
865 [ Debug ] Pop 62 from StkReturn , StkReturn.top = 81
866
867 [ Debug ] Push 63 into StkReturn
868
869 [ Debug ] Previous lac function execute VMTable[63] , VMTable[63] = 22
870
871 [ Debug ] Execute basic function - at VMTable[22]
872
873 [ Debug ] Pop 1 from StkData , StkData.top = 2
874
875 [ Debug ] Pop 2 from StkData , StkData.top = 2
876
877 [ Debug ] Push 1 into StkData
878
879 [ Debug ] Pop 63 from StkReturn , StkReturn.top = 81
880
```

```
881 [ Debug ] Push 64 into StkReturn
882 [ Debug ] Previous lac function execute VMTable[64], VMTable[64] = 28
884
885 [ Debug ] Execute basic function (fin) at VMTable[28]
886
887 [ Debug ] Pop 64 from StkReturn, StkReturn.top = 81
888
889 [ Debug ] Fonction (fin) executed, 64 is erased from StkReturn
890
891 [ Debug ] Pop 81 from StkReturn, StkReturn.top = 82
892
893 [ Debug ] Push 82 into StkReturn
894
895 [ Debug ] Previous lac function execute VMTable[82], VMTable[82] = 70
896
897 [ Debug ] Execute lac function at VMTable[70]
898
899 [ Debug ] Push 71 into StkReturn
900
901 [ Debug ] Previous lac function execute VMTable[71], VMTable[71] = 0
902
903 [ Debug ] Execute basic function dup at VMTable[0]
904
905 [ Debug ] Push 1 into StkData
906
907 [ Debug ] Pop 71 from StkReturn, StkReturn.top = 82
908
909 [ Debug ] Push 72 into StkReturn
910
911 [ Debug ] Previous lac function execute VMTable[72], VMTable[72] = 55
912
913 [ Debug ] Execute lac function at VMTable[55]
914
915 [ Debug ] Push 56 into StkReturn
916
917 [ Debug ] Previous lac function execute VMTable[56], VMTable[56] = 26
918
919 [ Debug ] Execute basic function (lit) at VMTable[26]
920
921 [ Debug ] Pop 56 from StkReturn, StkReturn.top = 72
922
923 [ Debug ] Push 57 into StkReturn
924
925 [ Debug ] Push 0 into StkData
926
927 [ Debug ] Fonction (lit) executed, 0 is read to StkData
928
929 [ Debug ] Pop 57 from StkReturn, StkReturn.top = 72
930
931 [ Debug ] Push 58 into StkReturn
932
933 [ Debug ] Previous lac function execute VMTable[58], VMTable[58] = 12
934
935 [ Debug ] Execute basic function = at VMTable[12]
936
937 [ Debug ] Pop 0 from StkData, StkData.top = 1
938
939 [ Debug ] Pop 1 from StkData, StkData.top = 1
940
941 [ Debug ] Push 0 into StkData
942
```

```
943 [ Debug ] Pop 58 from StkReturn, StkReturn.top = 72
944 [ Debug ] Push 59 into StkReturn
946
947 [ Debug ] Previous lac function execute VMTable[59], VMTable[59] = 28
948
949 [ Debug ] Execute basic function (fin) at VMTable[28]
950
951 [ Debug ] Pop 59 from StkReturn, StkReturn.top = 72
952
953 [ Debug ] Fonction (fin) executed, 59 is erased from StkReturn
954
955 [ Debug ] Pop 72 from StkReturn, StkReturn.top = 82
956
957 [ Debug ] Push 73 into StkReturn
958
959 [ Debug ] Previous lac function execute VMTable[73], VMTable[73] = 32
960
961 [ Debug ] Execute basic function if at VMTable[32]
962
963 [ Debug ] Pop 0 from StkData, StkData.top = 1
964
965 [ Debug ] Pop 73 from StkReturn, StkReturn.top = 82
966
967 [ Debug ] Push 79 into StkReturn
968
969 [ Debug ] Function If is not taken, jumps to 79 from 73
970
971 [ Debug ] Pop 79 from StkReturn, StkReturn.top = 82
972
973 [ Debug ] Push 80 into StkReturn
974
975 [ Debug ] Previous lac function execute VMTable[80], VMTable[80] = 0
976
977 [ Debug ] Execute basic function dup at VMTable[0]
978
979 [ Debug ] Push 1 into StkData
980
981 [ Debug ] Pop 80 from StkReturn, StkReturn.top = 82
982
983 [ Debug ] Push 81 into StkReturn
984
985 [ Debug ] Previous lac function execute VMTable[81], VMTable[81] = 60
986
987 [ Debug ] Execute lac function at VMTable[60]
988
989 [ Debug ] Push 61 into StkReturn
990
991 [ Debug ] Previous lac function execute VMTable[61], VMTable[61] = 26
992
993 [ Debug ] Execute basic function (lit) at VMTable[26]
994
995 [ Debug ] Pop 61 from StkReturn, StkReturn.top = 81
996
997 [ Debug ] Push 62 into StkReturn
998
999 [ Debug ] Push 1 into StkData
1000
1001 [ Debug ] Fonction (lit) executed, 1 is read to StkData
1002
1003 [ Debug ] Pop 62 from StkReturn, StkReturn.top = 81
1004
```

```
1005 [ Debug ] Push 63 into StkReturn
1006
1007 [ Debug ] Previous lac function execute VMTable[63] , VMTable[63] = 22
1008
1009 [ Debug ] Execute basic function - at VMTable[22]
1010
1011 [ Debug ] Pop 1 from StkData , StkData.top = 1
1012
1013 [ Debug ] Pop 1 from StkData , StkData.top = 1
1014
1015 [ Debug ] Push 0 into StkData
1016
1017 [ Debug ] Pop 63 from StkReturn , StkReturn.top = 81
1018
1019 [ Debug ] Push 64 into StkReturn
1020
1021 [ Debug ] Previous lac function execute VMTable[64] , VMTable[64] = 28
1022
1023 [ Debug ] Execute basic function (fin) at VMTable[28]
1024
1025 [ Debug ] Pop 64 from StkReturn , StkReturn.top = 81
1026
1027 [ Debug ] Fonction (fin) executed , 64 is erased from StkReturn
1028
1029 [ Debug ] Pop 81 from StkReturn , StkReturn.top = 82
1030
1031 [ Debug ] Push 82 into StkReturn
1032
1033 [ Debug ] Previous lac function execute VMTable[82] , VMTable[82] = 70
1034
1035 [ Debug ] Execute lac function at VMTable[70]
1036
1037 [ Debug ] Push 71 into StkReturn
1038
1039 [ Debug ] Previous lac function execute VMTable[71] , VMTable[71] = 0
1040
1041 [ Debug ] Execute basic function dup at VMTable[0]
1042
1043 [ Debug ] Push 0 into StkData
1044
1045 [ Debug ] Pop 71 from StkReturn , StkReturn.top = 82
1046
1047 [ Debug ] Push 72 into StkReturn
1048
1049 [ Debug ] Previous lac function execute VMTable[72] , VMTable[72] = 55
1050
1051 [ Debug ] Execute lac function at VMTable[55]
1052
1053 [ Debug ] Push 56 into StkReturn
1054
1055 [ Debug ] Previous lac function execute VMTable[56] , VMTable[56] = 26
1056
1057 [ Debug ] Execute basic function (lit) at VMTable[26]
1058
1059 [ Debug ] Pop 56 from StkReturn , StkReturn.top = 72
1060
1061 [ Debug ] Push 57 into StkReturn
1062
1063 [ Debug ] Push 0 into StkData
1064
1065 [ Debug ] Fonction (lit) executed , 0 is read to StkData
1066
```

```
1067 [ Debug ] Pop 57 from StkReturn , StkReturn.top = 72
1068 [ Debug ] Push 58 into StkReturn
1069
1070 [ Debug ] Previous lac function execute VMTable[58] , VMTable[58] = 12
1071
1072 [ Debug ] Execute basic function = at VMTable[12]
1073
1074 [ Debug ] Pop 0 from StkData , StkData.top = 0
1075
1076 [ Debug ] Pop 0 from StkData , StkData.top = 0
1077
1078 [ Debug ] Push 1 into StkData
1079
1080 [ Debug ] Pop 58 from StkReturn , StkReturn.top = 72
1081
1082 [ Debug ] Push 59 into StkReturn
1083
1084 [ Debug ] Previous lac function execute VMTable[59] , VMTable[59] = 28
1085
1086 [ Debug ] Execute basic function ( fin ) at VMTable[28]
1087
1088 [ Debug ] Pop 59 from StkReturn , StkReturn.top = 72
1089
1090 [ Debug ] Fonction ( fin ) executed , 59 is erased from StkReturn
1091
1092 [ Debug ] Pop 72 from StkReturn , StkReturn.top = 82
1093
1094 [ Debug ] Push 73 into StkReturn
1095
1096 [ Debug ] Previous lac function execute VMTable[73] , VMTable[73] = 32
1097
1098 [ Debug ] Execute basic function if at VMTable[32]
1099
1100 [ Debug ] Pop 1 from StkData , StkData.top = 0
1101
1102 [ Debug ] Pop 73 from StkReturn , StkReturn.top = 82
1103
1104 [ Debug ] Push 74 into StkReturn
1105
1106 [ Debug ] Function If is taken
1107
1108 [ Debug ] Pop 74 from StkReturn , StkReturn.top = 82
1109
1110 [ Debug ] Push 75 into StkReturn
1111
1112 [ Debug ] Previous lac function execute VMTable[75] , VMTable[75] = 2
1113
1114 [ Debug ] Execute basic function drop at VMTable[2]
1115
1116 [ Debug ] Pop 0 from StkData , StkData.top = 1
1117
1118 [ Debug ] Pop 75 from StkReturn , StkReturn.top = 82
1119
1120 [ Debug ] Push 76 into StkReturn
1121
1122 [ Debug ] Previous lac function execute VMTable[76] , VMTable[76] = 26
1123
1124 [ Debug ] Execute basic function ( lit ) at VMTable[26]
1125
1126 [ Debug ] Pop 76 from StkReturn , StkReturn.top = 82
1127
1128
```

```
129 [ Debug ] Push 77 into StkReturn
130
131 [ Debug ] Push 1 into StkData
132
133 [ Debug ] Fonction (lit) executed, 1 is read to StkData
134
135 [ Debug ] Pop 77 from StkReturn, StkReturn.top = 82
136
137 [ Debug ] Push 78 into StkReturn
138
139 [ Debug ] Previous lac function execute VMTable[78], VMTable[78] = 34
140
141 [ Debug ] Execute basic function else at VMTable[34]
142
143 [ Debug ] Function Else
144
145 [ Debug ] Pop 78 from StkReturn, StkReturn.top = 82
146
147 [ Debug ] Push 83 into StkReturn
148
149 [ Debug ] Unconditional jump to 83 from 78
150
151 [ Debug ] Pop 83 from StkReturn, StkReturn.top = 82
152
153 [ Debug ] Push 84 into StkReturn
154
155 [ Debug ] Previous lac function execute VMTable[84], VMTable[84] = 28
156
157 [ Debug ] Execute basic function (fin) at VMTable[28]
158
159 [ Debug ] Pop 84 from StkReturn, StkReturn.top = 82
160
161 [ Debug ] Fonction (fin) executed, 84 is erased from StkReturn
162
163 [ Debug ] Pop 82 from StkReturn, StkReturn.top = 82
164
165 [ Debug ] Push 83 into StkReturn
166
167 [ Debug ] Previous lac function execute VMTable[83], VMTable[83] = 18
168
169 [ Debug ] Execute basic function * at VMTable[18]
170
171 [ Debug ] Pop 1 from StkData, StkData.top = 1
172
173 [ Debug ] Pop 1 from StkData, StkData.top = 2
174
175 [ Debug ] Push 1 into StkData
176
177 [ Debug ] Pop 83 from StkReturn, StkReturn.top = 82
178
179 [ Debug ] Push 84 into StkReturn
180
181 [ Debug ] Previous lac function execute VMTable[84], VMTable[84] = 28
182
183 [ Debug ] Execute basic function (fin) at VMTable[28]
184
185 [ Debug ] Pop 84 from StkReturn, StkReturn.top = 82
186
187 [ Debug ] Fonction (fin) executed, 84 is erased from StkReturn
188
189 [ Debug ] Pop 82 from StkReturn, StkReturn.top = 82
190
```

```
191 [ Debug ] Push 83 into StkReturn
192 [ Debug ] Previous lac function execute VMTable[83] , VMTable[83] = 18
193 [ Debug ] Execute basic function * at VMTable[18]
194
195 [ Debug ] Pop 1 from StkData , StkData.top = 2
196
197 [ Debug ] Pop 2 from StkData , StkData.top = 3
198
199 [ Debug ] Push 2 into StkData
200
201 [ Debug ] Pop 83 from StkReturn , StkReturn.top = 82
202
203 [ Debug ] Push 84 into StkReturn
204
205 [ Debug ] Previous lac function execute VMTable[84] , VMTable[84] = 28
206
207 [ Debug ] Execute basic function (fin) at VMTable[28]
208
209 [ Debug ] Pop 84 from StkReturn , StkReturn.top = 82
210
211 [ Debug ] Fonction (fin) executed , 84 is erased from StkReturn
212
213 [ Debug ] Pop 82 from StkReturn , StkReturn.top = 82
214
215 [ Debug ] Push 83 into StkReturn
216
217 [ Debug ] Previous lac function execute VMTable[83] , VMTable[83] = 18
218
219 [ Debug ] Execute basic function * at VMTable[18]
220
221 [ Debug ] Pop 2 from StkData , StkData.top = 3
222
223 [ Debug ] Push 6 into StkData
224
225 [ Debug ] Pop 3 from StkData , StkData.top = 4
226
227 [ Debug ] Pop 83 from StkReturn , StkReturn.top = 82
228
229 [ Debug ] Push 84 into StkReturn
230
231 [ Debug ] Previous lac function execute VMTable[84] , VMTable[84] = 28
232
233 [ Debug ] Execute basic function (fin) at VMTable[28]
234
235 [ Debug ] Pop 84 from StkReturn , StkReturn.top = 82
236
237 [ Debug ] Fonction (fin) executed , 84 is erased from StkReturn
238
239 [ Debug ] Pop 82 from StkReturn , StkReturn.top = 82
240
241 [ Debug ] Push 83 into StkReturn
242
243 [ Debug ] Previous lac function execute VMTable[83] , VMTable[83] = 18
244
245 [ Debug ] Execute basic function * at VMTable[18]
246
247 [ Debug ] Pop 6 from StkData , StkData.top = 4
248
249 [ Debug ] Pop 4 from StkData , StkData.top = 5
250
251 [ Debug ] Pop 4 from StkData , StkData.top = 5
252
```

```
1253 [ Debug ] Push 24 into StkData
1254 [ Debug ] Pop 83 from StkReturn, StkReturn.top = 82
1256
1257 [ Debug ] Push 84 into StkReturn
1258
1259 [ Debug ] Previous lac function execute VMTable[84], VMTable[84] = 28
1260
1261 [ Debug ] Execute basic function (fin) at VMTable[28]
1262
1263 [ Debug ] Pop 84 from StkReturn, StkReturn.top = 82
1264
1265 [ Debug ] Fonction (fin) executed, 84 is erased from StkReturn
1266
1267 [ Debug ] Pop 82 from StkReturn, StkReturn.top = 82
1268
1269 [ Debug ] Push 83 into StkReturn
1270
1271 [ Debug ] Previous lac function execute VMTable[83], VMTable[83] = 18
1272
1273 [ Debug ] Execute basic function * at VMTable[18]
1274
1275 [ Debug ] Pop 24 from StkData, StkData.top = 5
1276
1277 [ Debug ] Pop 5 from StkData, StkData.top = 6
1278
1279 [ Debug ] Push 120 into StkData
1280
1281 [ Debug ] Pop 83 from StkReturn, StkReturn.top = 82
1282
1283 [ Debug ] Push 84 into StkReturn
1284
1285 [ Debug ] Previous lac function execute VMTable[84], VMTable[84] = 28
1286
1287 [ Debug ] Execute basic function (fin) at VMTable[28]
1288
1289 [ Debug ] Pop 84 from StkReturn, StkReturn.top = 82
1290
1291 [ Debug ] Fonction (fin) executed, 84 is erased from StkReturn
1292
1293 [ Debug ] Pop 82 from StkReturn, StkReturn.top = 125
1294
1295 [ Debug ] Push 83 into StkReturn
1296
1297 [ Debug ] Previous lac function execute VMTable[83], VMTable[83] = 18
1298
1299 [ Debug ] Execute basic function * at VMTable[18]
1300
1301 [ Debug ] Pop 120 from StkData, StkData.top = 6
1302
1303 [ Debug ] Pop 6 from StkData, StkData.top = 52
1304
1305 [ Debug ] Push 720 into StkData
1306
1307 [ Debug ] Pop 83 from StkReturn, StkReturn.top = 125
1308
1309 [ Debug ] Push 84 into StkReturn
1310
1311 [ Debug ] Previous lac function execute VMTable[84], VMTable[84] = 28
1312
1313 [ Debug ] Execute basic function (fin) at VMTable[28]
1314
```

```
1315 [ Debug ] Pop 84 from StkReturn , StkReturn.top = 125
1316
1317 [ Debug ] Fonction (fin) executed , 84 is erased from StkReturn
1318
1319 [ Debug ] Pop 125 from StkReturn , StkReturn.top = N/A
1320
1321 [ Debug ] Push 126 into StkReturn
1322
1323 [ Debug ] Previous lac function execute VMTable[126] , VMTable[126] = 6
1324
1325 [ Debug ] Execute basic function . at VMTable[6]
1326
1327 [ Debug ] Pop 720 from StkData , StkData.top = 52
1328 720
1329 [ Debug ] Pop 126 from StkReturn , StkReturn.top = N/A
1330
1331 [ Debug ] Push 127 into StkReturn
1332
1333 [ Debug ] Previous lac function execute VMTable[127] , VMTable[127] = 65
1334
1335 [ Debug ] Execute lac function at VMTable[65]
1336
1337 [ Debug ] Push 66 into StkReturn
1338
1339 [ Debug ] Previous lac function execute VMTable[66] , VMTable[66] = 26
1340
1341 [ Debug ] Execute basic function (lit) at VMTable[26]
1342
1343 [ Debug ] Pop 66 from StkReturn , StkReturn.top = 127
1344
1345 [ Debug ] Push 67 into StkReturn
1346
1347 [ Debug ] Push 1 into StkData
1348
1349 [ Debug ] Fonction (lit) executed , 1 is read to StkData
1350
1351 [ Debug ] Pop 67 from StkReturn , StkReturn.top = 127
1352
1353 [ Debug ] Push 68 into StkReturn
1354
1355 [ Debug ] Previous lac function execute VMTable[68] , VMTable[68] = 20
1356
1357 [ Debug ] Execute basic function + at VMTable[20]
1358
1359 [ Debug ] Pop 1 from StkData , StkData.top = 52
1360
1361 [ Debug ] Pop 52 from StkData , StkData.top = N/A
1362
1363 [ Debug ] Push 53 into StkData
1364
1365 [ Debug ] Pop 68 from StkReturn , StkReturn.top = 127
1366
1367 [ Debug ] Push 69 into StkReturn
1368
1369 [ Debug ] Previous lac function execute VMTable[69] , VMTable[69] = 28
1370
1371 [ Debug ] Execute basic function (fin) at VMTable[28]
1372
1373 [ Debug ] Pop 69 from StkReturn , StkReturn.top = 127
1374
1375 [ Debug ] Fonction (fin) executed , 69 is erased from StkReturn
1376
```

```
1377 [ Debug ] Pop 127 from StkReturn, StkReturn.top = N/A
1378 [ Debug ] Push 128 into StkReturn
1380
1381 [ Debug ] Previous lac function execute VMTable[128], VMTable[128] = 28
1382 [ Debug ] Execute basic function (fin) at VMTable[28]
1384
1385 [ Debug ] Pop 128 from StkReturn, StkReturn.top = N/A
1386
1387 [ Debug ] Fonction (fin) executed, 128 is erased from StkReturn
1388
1389 [ Info ] Execution finished
```