Rapport Livrable 3 - Assembleur PYTHON

Paco LARDY–NUGUES, Abdourahmane MBAYE,
Idriss ABDOULWAHAB, Lorenzo AZERINE
SICOM

22/10/2023

Lien vers dépôt **Gitlab**.

1 Tâches

1.1 Début livrable 3 + Corrections bugs (Responsable : Paco LARDY–NUGUES)

Travail réalisé :

- Réalisation de fichiers tests *.pys* pour des futurs tests.

- Correction de Warnings.

- Réalisation de fichiers d'exemple source en *.pys* (dans le fichier *tests/python_test_folder*)

- **Livrable 3 :** Fonctions suivantes :

Code Listing 1 – Fonctions de bases livrable 3

```
1
2   int lexem_type_strict(lexem_t lexem, char * type_t);
3   lexem_t lexem_peek(list_t *lexems);
4   list_t lexem_advance( list_t *lexems );
5   int next_lexem_is( list_t *lexems, char *type );
6   void print_parse_error( char *msg, list_t *lexems );
```

Avancés :

- Les fonctions sont réalisées et testés avec *lexem_test*.

1.2 Début livrable 3 + Création du type *lexem_type_t* | Mise à jour de la fonction *lex_read* (Responsable : Lorenzo AZERINE)

Travail réalisé :

-Création du type : lexem_type_t pour stocker les types de lexèmes, ce type contient le type du lexem, sa valeur sous forme de chaine de caractère et sa valeur sous forme de liste de char_group défini au livrable 1. Mise à jour de la fonction lex_read en conséquence, la fonction n'utilise plus le type lexem_t mais bien lexem_type_t.

Code Listing 2 – Lexem_type

```
struct lexem_type {
        char *type;
        char *value;
        list_t l_value ;
    };

    typedef struct lexem_type *lexem_type_t;
    lexem_type_t * lex_read(char *filename);
    int lexem_type_print( void *_lex );
```

- Réalisation des tests asssociés.

- **Livrable 3 :** Implémentation de la fonction parse_if qui prends en entrée la liste de lexèmes lus dans le fichier *.pys* et un l'indice du maillon actuel de la liste des lexèmes. Elle renvoie un entier en fonction du résultat de l'opération booléenne sous jacente à l'instruction *if* : 0 si False et 1 si True.

1.3 Conception des tests de la fonction lex (Responsable : Idriss ABDOULWAHAB)

Travail réalisé :

- Conception des tests unitaires et d'intégrations de la fonction *lex* afin de peaufiner et finir le livrable 2.

1.4 Conception de la fonction *lex* (Responsable : Abdourahmane MBAYE)

Travail réalisé :

- Conception de la fonction *Lexer* prenant en paramètre un fichier configuration *.conf* et un fichier source *.pys* et retournant la liste de *lexèmes* présents avec pour chacun la ligne et colonne correspondante.

1.5 Livrable 3 : Conception du langage EBNF python (Responsables : Paco LARDY–NUGUES et Lorenzo AZERINE)

- Conception du langage **EBNF** python avec lequel nous allons construire notre *Abstract Syntax Tree* afin de construire notre *parser*.

Code Listing 3 – Langage EBNF Python

```
<programme> ::= <instruction>*

<instruction> ::= <affectation>
                | <condition>
                | <boucle>
                | <fonction>
                | <expression>

<affectation> ::= <identifier> {op::assignment} <expression>

<condition> ::= "if" <expression>":"<programme>
                ("elif"<expression>":"<programme>)*
                ("else" ":" <programme>)?

<boucle> ::= "while" <expression> ":" <programme>
           | "for" <identifier> "in" <expression> ":" <programme>

<fonction> ::= "def" <identifier> "(" <parametres>? ")" ":" <programme>

<expression> ::= <arith-expr>
               | <bool-expr>
               | <list>
               | <str>

<parametres> ::= <identifier> ( "," <identifier> )*

<arith-expr> ::= <term> ( ( {op::sum::plus} | {op::sum::minus} ) <term> )*

<term> ::= <s-factor> ( ( {op::prod::mul} | {op::prod::div} ) <s-factor> )*

<s-factor> ::= [ {op::sum::plus} | {op::sum::minus} ] <factor>

<factor> ::= <number>
           | <identifier>
           | {paren::left} <arith-expr> {paren::right}
           | <bool-expr>

<list> ::= "[" (<element> ","?)* "]"

<element> ::= <identifier> | <arith-expr> | <str> | <bool-expr> | <list>

<str> ::= '"' {string} '"'

<bool-expr> ::= <bool-term> ( {op::bool-or} <bool-term> )*

<bool-term> ::= <bool-factor> ( {op::bool-and} <bool-factor> )*

<bool-factor> ::= [ {op::bool-not} ] <bool-primary>

<bool-primary> ::= "true" | "false"
                 | <comparison>

<comparison> ::= <arith-expr> {op::compare} <arith-expr>

<identifier> ::= {identifier}

<number> ::= {number}

<paren::left> ::= "("
<paren::right> ::= ")"
```

```
{string} = [^"]*

{op::sum::plus}::="+"
{op::sum::minus}::="-"
{op::prod::mul}::="*"
{op::prod::div}::="/"
{op::assignment}::="="
{op::bool-or}::="or"
{op::bool-and}::="and"
{op::bool-not}::="not"
{op::compare}::="=="|"$\!$="|">"|"<"|">="|"<="
```

Avancés :

- Nous nous sommes rendu compte à la fin que ce langage avait déjà était écrit dans le **sujet**, par conséquent nous utiliserons celui donné. Nous avons donc commencer à construire notre *py_obj* ainsi que les fonctions qui en découlent, en ne sachant pas l'existence du lanngage **EBNF** donné dans le sujet, nous allons donc refaire.

2 Conclusion

Le livrable 2 est pratiquement terminé (avec plus ou moins quelques soucis mineurs), et le livrable 3 avance. Nous n'avons pas tenu nos objectifs de terminer le livrable 3, nous redoublerons d'efforts.