TP3

Generated by Doxygen 1.8.13

Contents

Chapter 1

Troisième exercice du TP IL2

Il s'agit de réaliser l'analyseur syntaxique pour le langage JSON

mode d'emploi

1ere étape : récupérer le matériel initial :

```
git clone https://gitlab.com/nicolas.monmarche/tp_il2_ex3.git
```

2ieme étape créer votre branche (remplacer nom-binome par ce qui vous concerne) :

```
git branch "nom-binome"
git checkout "Nom-binome"
```

3ieme étape transférer votre commit initial

```
git commit -m "notre premiere version"
git push --set-upstream origin "Nom-binome"
```

n-étape

à chaque fois que vous souhaitez tranférer votre travail :

```
git commit -m "explications" fichiers-modifiés
git push
```

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 _json_array Struct Reference

`#include <json_tree.h>`

Collaboration diagram for _json_array:

Data Fields

- JsonValueContainer ** elements
- int size

4.1.1 Field Documentation

4.1.1.1 elements

`JsonValueContainer** _json_array::elements`

liste des elements

4.1.1.2 size

`int _json_array::size`

nombre d'elements

The documentation for this struct was generated from the following file:

- json_tree.h

4.2 _json_object Struct Reference

`#include <json_tree.h>`

Collaboration diagram for _json_object:

Data Fields

- JsonPair ** members
- int size

4.2.1 Field Documentation

4.2.1.1 members

`JsonPair** _json_object::members`

liste des membres

4.2.1.2 size

`int _json_object::size`

nombre de membre

The documentation for this struct was generated from the following file:

- json_tree.h

4.3 _json_pair Struct Reference

`#include <json_tree.h>`

Collaboration diagram for _json_pair:

Data Fields

- char * string
- struct _json_value_container * value

4.3.1 Field Documentation

4.3.1.1 string

```
char* _json_pair::string
```

name

4.3.1.2 value

```
struct _json_value_container* _json_pair::value
```

value

The documentation for this struct was generated from the following file:

- json_tree.h

4.4 _json_value_container Struct Reference

```
#include <json_tree.h>
```

Collaboration diagram for _json_value_container:

Data Fields

- ValueType type
- JsonValue value

4.4.1 Field Documentation

4.4.1.1 type

```
ValueType _json_value_container::type
```

type de valeur

4.4.1.2 value

```
JsonValue _json_value_container::value
```

The documentation for this struct was generated from the following file:

- json_tree.h

4.5 JsonArray Struct Reference

pour les tableau "array"

`#include <json_tree.h>`

4.5.1 Detailed Description

pour les tableau "array"

The documentation for this struct was generated from the following file:

- json_tree.h

4.6 JsonObject Struct Reference

pour stoker un objet JSON

`#include <json_tree.h>`

4.6.1 Detailed Description

pour stoker un objet JSON

The documentation for this struct was generated from the following file:

- json_tree.h

4.7 JsonPair Struct Reference

pour stocker une "pair" string : value

`#include <json_tree.h>`

4.7.1 Detailed Description

pour stocker une "pair" string : value

The documentation for this struct was generated from the following file:

- json_tree.h

4.8 JsonValue Union Reference

pour stocker "value"

`#include <json_tree.h>`

Collaboration diagram for JsonValue:

Data Fields

- char * string
- int integer
- float real
- struct _json_object * object
- struct _json_array * array
- int constant

4.8.1 Detailed Description

pour stocker "value"

4.8.2 Field Documentation

4.8.2.1 array

```
struct _json_array* JsonValue::array
```

4.8.2.2 constant

```
int JsonValue::constant
```

4.8.2.3 integer

```
int JsonValue::integer
```

4.8.2.4 object

```
struct _json_object* JsonValue::object
```

4.8.2.5 real

```
float JsonValue::real
```

4.8.2.6 string

```
char* JsonValue::string
```

The documentation for this union was generated from the following file:

- json_tree.h

4.9 JsonValueContainer Struct Reference

pour stocker "value" et son type

```
#include <json_tree.h>
```

4.9.1 Detailed Description

pour stocker "value" et son type

The documentation for this struct was generated from the following file:

- json_tree.h

4.10 TIntPile Struct Reference

structure contenant une pile d'entiers

```
#include <pile.h>
```

Data Fields

- int * data
- int indexSommet
- int size

4.10.1 Detailed Description

structure contenant une pile d'entiers

pile contenant des entiers

4.10.2 Field Documentation

4.10.2.1 data

```
int* TIntPile::data
```

tableau d'entiers representant la pile

4.10.2.2 indexSommet

```
int TIntPile::indexSommet
```

indice du sommet

4.10.2.3 size

```
int TIntPile::size
```

taille en mémoire de la pile

The documentation for this struct was generated from the following file:

- pile.h

4.11 TLex Struct Reference

structure contenant tous les parametres/donnees pour l'analyse lexicale

```
#include <tp2_lex.h>
```

Collaboration diagram for TLex:

Data Fields

- char * data
- char * startPos
- int nbLignes
- TSymbole * tableSymboles
- int nbSymboles
- int tailleTableSymboles

4.11.1 Detailed Description

structure contenant tous les parametres/donnees pour l'analyse lexicale

4.11.2 Field Documentation

4.11.2.1 data

```
char* TLex::data
```

chaine a parcourir

4.11.2.2 nbLignes

```
int TLex::nbLignes
```

nb de lignes analysees

4.11.2.3 nbSymboles

```
int TLex::nbSymboles
```

nb de symboles stockes dans tableSymboles

4.11.2.4 startPos

```
char* TLex::startPos
```

position de depart pour la prochaine analyse

4.11.2.5 tableSymboles

```
TSymbole* TLex::tableSymboles
```

tableau des symboles : chaines/entier/reel

4.11.2.6 tailleTableSymboles

```
int TLex::tailleTableSymboles
```

taille memoire du tableau tableSymboles

The documentation for this struct was generated from the following file:

- tp2_lex.h

4.12 TSymbole Union Reference

union permettant de manipuler un entier/reel/chaine pour la table des symboles

```
#include <tp2_lex.h>
```

Data Fields

- int type
- union {
  int entier
  float reel
  char * chaine
  } val

4.12.1 Detailed Description

union permettant de manipuler un entier/reel/chaine pour la table des symboles

4.12.2 Field Documentation

4.12.2.1 chaine

```
char* TSymbole::chaine
```

4.12.2.2 entier

```
int TSymbole::entier
```

4.12.2.3 reel

```
float TSymbole::reel
```

4.12.2.4 type

```
int TSymbole::type
```

l'un des 3 types suivants : JSON_STRING/JSON_INT_NUMBER/JSON_REAL_NUMBER

4.12.2.5 val

```
union { ... } TSymbole::val
```

valeur associer a un element de la table des symboles

The documentation for this union was generated from the following file:

- tp2_lex.h

4.13 TVoidPile Struct Reference

structure contenant une pile de pointeur void *

```
#include <pile.h>
```

Data Fields

- void ** data
- int indexSommet
- int size

4.13.1 Detailed Description

structure contenant une pile de pointeur void *

pile contenant des pointeurs sur des objets

4.13.2 Field Documentation

4.13.2.1 data

```
void** TVoidPile::data
```

tableau de pointeur void *

4.13.2.2 indexSommet

```
int TVoidPile::indexSommet
```

indice du sommet

4.13.2.3 size

```
int TVoidPile::size
```

taille en memoire de la pile

The documentation for this struct was generated from the following file:

- pile.h

Chapter 5

File Documentation

5.1 json_tree.c File Reference

implemente la mise en memoire d'un arbre json

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "json_tree.h"
```

Include dependency graph for json_tree.c:

5.2 json_tree.h File Reference

implemente la mise en memoire d'un arbre json

This graph shows which files directly or indirectly include this file:

Data Structures

- struct _json_pair
- union JsonValue

  pour stocker "value"
- struct _json_value_container
- struct _json_object
- struct _json_array

Macros

- #define CSTE_JSON_NULL 2
- #define CSTE_JSON_TRUE 1
- #define CSTE_JSON_FALSE 0

Typedefs

- typedef struct _json_pair JsonPair
- typedef enum _value_type ValueType
- typedef struct _json_value_container JsonValueContainer
- typedef struct _json_object JsonObject
- typedef struct _json_array JsonArray

Enumerations

- enum _value_type {
  string, integer, real, object,
  array, constant }

Functions

- JsonArray * CreateJsonArray ()

  creation en memoire d'un JsonArray
- int InsertJsonArray (JsonArray *_array, JsonValueContainer *_valueContainer, const unsigned int _position)
- int DeleteJsonArray (JsonArray **_array)
- char * PrintJsonArray (const JsonArray *_array)
- char * PrintDotJsonArray (const JsonArray *_array)
- JsonObject * CreateJsonObject ()

  construit un JsonObject vide
- int InsertJsonObject (JsonObject *_object, JsonPair *_pair)
- JsonValueContainer * GetJsonValueContainer (const JsonObject *_object, const char *_string)
- int DeleteJsonObject (JsonObject **_object)
- char * PrintJsonObject (const JsonObject *_object)
- char * PrintDotJsonObject (const JsonObject *_object, int _id, int _idParent)
- JsonValueContainer * CreateJsonValueContainer ()

  construit un JsonValueContainer vide
- void UpdateJsonValueContainer (JsonValueContainer *_valueContainer, ValueType _type, JsonValue _↩ value)
- int DeleteJsonValueContainer (JsonValueContainer **_valueContainer)
- char * PrintJsonValueContainer (const JsonValueContainer *_valueContainer)
- JsonPair * CreateJsonPair ()
- void UpdateJsonPair (JsonPair *_pair, char *_string, JsonValueContainer *_value)
- int DeleteJsonPair (JsonPair **_pair)
- char * PrintJsonPair (const JsonPair *_pair)

5.2.1 Detailed Description

implemente la mise en memoire d'un arbre json

Author

NM

Version

0.1

Date

06.12.2014

5.2.2 Macro Definition Documentation

5.2.2.1 CSTE_JSON_FALSE

```
#define CSTE_JSON_FALSE 0
```

5.2.2.2 CSTE_JSON_NULL

```
#define CSTE_JSON_NULL 2
```

5.2.2.3 CSTE_JSON_TRUE

```
#define CSTE_JSON_TRUE 1
```

5.2.3 Typedef Documentation

5.2.3.1 JsonArray

```
typedef struct _json_array JsonArray
```

5.2.3.2 JsonObject

```
typedef struct _json_object JsonObject
```

5.2.3.3 JsonPair

```
typedef struct _json_pair JsonPair
```

5.2.3.4 JsonValueContainer

```
typedef struct _json_value_container JsonValueContainer
```

5.2.3.5 ValueType

```
typedef enum _value_type ValueType
```

5.2.4 Enumeration Type Documentation

5.2.4.1 _value_type

```
enum _value_type
```

Enumerator

string	
integer	
real	
object	
array	
constant	pour true, false et null

5.2.5 Function Documentation

5.2.5.1 CreateJsonArray()

```
JsonArray* CreateJsonArray ( )
```

creation en memoire d'un JsonArray

Returns

JsonArray*

5.2.5.2 CreateJsonObject()

```
JsonObject* CreateJsonObject ( )
```

construit un JsonObject vide

Returns

JsonObject *

5.2.5.3 CreateJsonPair()

```
JsonPair* CreateJsonPair ( )
```

Parameters

5.2.5.4 CreateJsonValueContainer()

```
JsonValueContainer* CreateJsonValueContainer ( )
```

construit un JsonValueContainer vide

Parameters

5.2.5.5 DeleteJsonArray()

```
int DeleteJsonArray (
            JsonArray ** _array )
```

5.2.5.6 DeleteJsonObject()

```
int DeleteJsonObject (
            JsonObject ** _object )
```

5.2.5.7 DeleteJsonPair()

```
int DeleteJsonPair (
            JsonPair ** _pair )
```

5.2.5.8 DeleteJsonValueContainer()

```
int DeleteJsonValueContainer (
            JsonValueContainer ** _valueContainer )
```

5.2.5.9 GetJsonValueContainer()

```
JsonValueContainer* GetJsonValueContainer (
            const JsonObject * _object,
            const char * _string )
```

5.2.5.10 InsertJsonArray()

```
int InsertJsonArray (
            JsonArray * _array,
            JsonValueContainer * _valueContainer,
            const unsigned int _position )
```

5.2.5.11 InsertJsonObject()

```
int InsertJsonObject (
            JsonObject * _object,
            JsonPair * _pair )
```

5.2.5.12 PrintDotJsonArray()

```
char* PrintDotJsonArray (
            const JsonArray * _array )
```

5.2.5.13 PrintDotJsonObject()

```
char* PrintDotJsonObject (
            const JsonObject * _object,
            int _id,
            int _idParent )
```

5.2.5.14 PrintJsonArray()

```
char* PrintJsonArray (
            const JsonArray * _array )
```

5.2.5.15 PrintJsonObject()

```
char* PrintJsonObject (
            const JsonObject * _object )
```

5.2.5.16 PrintJsonPair()

```
char* PrintJsonPair (
            const JsonPair * _pair )
```

5.2.5.17 PrintJsonValueContainer()

```
char* PrintJsonValueContainer (
            const JsonValueContainer * _valueContainer )
```

5.2.5.18 UpdateJsonPair()

```
void UpdateJsonPair (
            JsonPair * _pair,
            char * _string,
            JsonValueContainer * _value )
```

5.2.5.19 UpdateJsonValueContainer()

```
void UpdateJsonValueContainer (
            JsonValueContainer * _valueContainer,
            ValueType _type,
            JsonValue _value )
```

5.3 pile.c File Reference

gestion d'une pile

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "pile.h"
```

Include dependency graph for pile.c:

Macros

- #define _DEFAULT_PILE_SIZE 15

Functions

- TIntPile * initIntPile ()
- void deleteIntPile (TIntPile **_pile)

  efface la memoire occupe par la pile
- void printIntPile (TIntPile *_pile)

  affichage du contenu d'une pile
- void empilerInt (TIntPile *_pile, int _val)

  empiler un entier sur la pile, si la zone memoire reservee n'est pas suffisante, celle-ci est etendue
- int depilerInt (TIntPile *_pile)

  dépiler un entier
- int sommetInt (TIntPile *_pile)

  renvoie la valeur du sommet (sans depiler)
- TVoidPile * initVoidPile ()

  *fonction qui initialise une pile de pointeur void **
- void deleteVoidPile (TVoidPile **_pile)

  libere la memoire occupee par la pile
- void printVoidPile (TVoidPile *_pile)

  affichage de la pile (comme on ne connait pas les type des elements empiles, on affiche les adresses de tous les pointeurs empiles)
- void empilerVoid (TVoidPile *_pile, void *_val)

  *empile un void **
- void * depilerVoid (TVoidPile *_pile)

  *dépiler un élément de type void **
- void * sommetVoid (TVoidPile *_pile)

  *obtenir la valeur du sommet de type void **

5.3.1 Detailed Description

gestion d'une pile

Author

GILBERT Dorian et DHONDT Matheo

Version

0.2

Date

11/01/2023

5.3.2 Macro Definition Documentation

5.3.2.1 _DEFAULT_PILE_SIZE

```
#define _DEFAULT_PILE_SIZE 15
```

constante pour la taille par defaut d'une pile (utilise pour la reservation memoire)

5.3.3 Function Documentation

5.3.3.1 deleteIntPile()

```
void deleteIntPile (
            TIntPile ** _pile )
```

efface la memoire occupe par la pile

Parameters

5.3.3.2 deleteVoidPile()

```
void deleteVoidPile (
```

```
            TVoidPile ** _pile )
```

libere la memoire occupee par la pile

Parameters

in	*_pile*	: adresse du pointeur sur la pile a liberer

Returns

neant

5.3.3.3 depilerInt()

```
int depilerInt (
            TIntPile * _pile )
```

dépiler un entier

Parameters

in	*_pile*	: la pile a depiler

Returns

l'entier en sommet de pile (0 si la pile est vide)

5.3.3.4 depilerVoid()

```
void * depilerVoid (
            TVoidPile * _pile )
```

dépiler un élément de type void *

Parameters

in	*_pile*	: pile a utiliser

Returns

pointeur sur void (0 si la pile est vide)

5.3.3.5 empilerInt()

```
void empilerInt (
            TIntPile * _pile,
            int _val )
```

empiler un entier sur la pile, si la zone memoire reservee n'est pas suffisante, celle-ci est etendue

Parameters

5.3.3.6 empilerVoid()

```
void empilerVoid (
            TVoidPile * _pile,
            void * _val )
```

empile un void *

Parameters

5.3.3.7 initIntPile()

```
TIntPile* initIntPile ( )
```

5.3.3.8 initVoidPile()

```
TVoidPile * initVoidPile ( )
```

fonction qui initialise une pile de pointeur void *

pile de void * ----------

Returns

pointeur sur une pile TVoidPile

5.3.3.9 printIntPile()

```
void printIntPile (
            TIntPile * _pile )
```

affichage du contenu d'une pile

Parameters

in	*_pile*	: la pile a afficher

Returns

neant

5.3.3.10 printVoidPile()

```
void printVoidPile (
            TVoidPile * _pile )
```

affichage de la pile (comme on ne connait pas les type des elements empiles, on affiche les adresses de tous les pointeurs empiles)

Parameters

in	*_pile*	: pile a afficher

Returns

neant

5.3.3.11 sommetInt()

```
int sommetInt (
            TIntPile * _pile )
```

renvoie la valeur du sommet (sans depiler)

Parameters

in	*_pile*	: la pile a utiliser

Returns

l'entier en sommet de pile (0 si la pile est vide)

5.3.3.12 sommetVoid()

```
void * sommetVoid (
            TVoidPile * _pile )
```

obtenir la valeur du sommet de type void *

Parameters

in	*_pile*	: pile a utiliser pour lire le sommet

Returns

la valeur void * du sommet (0 si la pile est vide)

5.4 pile.h File Reference

gestion d'une pile

This graph shows which files directly or indirectly include this file:

Data Structures

- struct TIntPile
  structure contenant une pile d'entiers
- struct TVoidPile
  *structure contenant une pile de pointeur void **

Functions

- TIntPile * initIntPile ()
- void deleteIntPile (TIntPile **_pile)
  efface la memoire occupe par la pile
- void printIntPile (TIntPile *_pile)
  affichage du contenu d'une pile
- void empilerInt (TIntPile *_pile, int _val)
  empiler un entier sur la pile, si la zone memoire reservee n'est pas suffisante, celle-ci est etendue
- int depilerInt (TIntPile *_pile)
  dépiler un entier
- int sommetInt (TIntPile *_pile)
  renvoie la valeur du sommet (sans depiler)
- TVoidPile * initVoidPile ()
  *fonction qui initialise une pile de pointeur void **
- void deleteVoidPile (TVoidPile **_pile)
  libere la memoire occupee par la pile
- void printVoidPile (TVoidPile *_pile)
  affichage de la pile (comme on ne connait pas les type des elements empiles, on affiche les adresses de tous les pointeurs empiles)
- void empilerVoid (TVoidPile *_pile, void *_val)
  *empile un void **
- void * depilerVoid (TVoidPile *_pile)
  *dépiler un élément de type void **
- void * sommetVoid (TVoidPile *_pile)
  *obtenir la valeur du sommet de type void **

5.4.1 Detailed Description

gestion d'une pile

Author

NM

Version

0.1

Date

11/12/2015

5.4.2 Function Documentation

5.4.2.1 deleteIntPile()

```
void deleteIntPile (
            TIntPile ** _pile )
```

efface la memoire occupe par la pile

Parameters

5.4.2.2 deleteVoidPile()

```
void deleteVoidPile (
            TVoidPile ** _pile )
```

libere la memoire occupee par la pile

Parameters

in	*_pile*	: adresse du pointeur sur la pile a liberer

Returns

neant

5.4.2.3 depilerInt()

```
int depilerInt (
            TIntPile * _pile )
```

dépiler un entier

Parameters

in	*_pile*	: la pile a depiler

Returns

l'entier en sommet de pile (0 si la pile est vide)

5.4.2.4 depilerVoid()

```
void* depilerVoid (
            TVoidPile * _pile )
```

dépiler un élément de type void *

Parameters

in	*_pile*	: pile a utiliser

Returns

pointeur sur void (0 si la pile est vide)

5.4.2.5 empilerInt()

```
void empilerInt (
            TIntPile * _pile,
            int _val )
```

empiler un entier sur la pile, si la zone memoire reservee n'est pas suffisante, celle-ci est etendue

Parameters

5.4.2.6 empilerVoid()

```
void empilerVoid (
            TVoidPile * _pile,
            void * _val )
```

empile un void *

Parameters

5.4.2.7 initIntPile()

```
TIntPile* initIntPile ( )
```

5.4.2.8 initVoidPile()

```
TVoidPile* initVoidPile ( )
```

fonction qui initialise une pile de pointeur void *

pile de void * ----------

Returns

pointeur sur une pile TVoidPile

5.4.2.9 printIntPile()

```
void printIntPile (
            TIntPile * _pile )
```

affichage du contenu d'une pile

Parameters

in	*_pile*	: la pile a afficher

Returns

neant

5.4.2.10 printVoidPile()

```
void printVoidPile (
            TVoidPile * _pile )
```

affichage de la pile (comme on ne connait pas les type des elements empiles, on affiche les adresses de tous les pointeurs empiles)

Parameters

in	*_pile*	: pile a afficher

Returns

neant

5.4.2.11 sommetInt()

```
int sommetInt (
            TIntPile * _pile )
```

renvoie la valeur du sommet (sans depiler)

Parameters

in	*_pile*	: la pile a utiliser

Returns

l'entier en sommet de pile (0 si la pile est vide)

5.4.2.12 sommetVoid()

```
void* sommetVoid (
            TVoidPile * _pile )
```

obtenir la valeur du sommet de type void *

Parameters

in	*_pile*	: pile a utiliser pour lire le sommet

Returns

la valeur void * du sommet (0 si la pile est vide)

5.5 README.md File Reference

5.6 tp2_lex.c File Reference

analyseur lexical pour le langage JSON

```
#include "tp2_lex.h"
```

Include dependency graph for tp2_lex.c:

Functions

- int isSep (const char _symb)
  fonction qui teste si un symbole fait partie des separateurs
- TLex * initLexData (char *_data)
  fonction qui reserve la memoire et initialise les donnees pour l'analyseur lexical
- void deleteLexData (TLex **_lexData)
  fonction qui supprime de la memoire les donnees pour l'analyseur lexical
- void printLexData (TLex *_lexData)
  fonction qui affiche les donnees pour l'analyseur lexical
- void addIntSymbolToLexData (TLex *_lexData, const int _val)
  fonction qui ajoute un symbole entier a la table des symboles
- void addRealSymbolToLexData (TLex *_lexData, const float _val)
  fonction qui ajoute un symbole reel a la table des symboles
- void addStringSymbolToLexData (TLex *_lexData, char *_val)
  fonction qui ajoute une chaine de caracteres a la table des symboles
- int lex (TLex *_lexData)

5.6.1 Detailed Description

analyseur lexical pour le langage JSON

Author

GILBERT Dorian et DHONDT Matheo

Version

0.1

Date

04/01/2023

5.6.2 Function Documentation

5.6.2.1 addIntSymbolToLexData()

```
void addIntSymbolToLexData (
            TLex * _lexData,
            const int _val )
```

fonction qui ajoute un symbole entier a la table des symboles

Parameters

in,out	*_lexData*	donnees de l'analyseur lexical
in	*_val*	valeur entiere a ajouter

Returns

neant

5.6.2.2 addRealSymbolToLexData()

```
void addRealSymbolToLexData (
            TLex * _lexData,
            const float _val )
```

fonction qui ajoute un symbole reel a la table des symboles

Parameters

in,out	*_lexData*	donnees de l'analyseur lexical
in	*_val*	valeur reelle a ajouter

5.6.2.3 addStringSymbolToLexData()

```
void addStringSymbolToLexData (
            TLex * _lexData,
            char * _val )
```

fonction qui ajoute une chaine de caracteres a la table des symboles

Parameters

in,out	*_lexData*	donnees de l'analyseur lexical
in	*_val*	chaine a ajouter

5.6.2.4 deleteLexData()

```
void deleteLexData (
            TLex ** _lexData )
```

fonction qui supprime de la memoire les donnees pour l'analyseur lexical

Parameters

in,out	*_lexData*	donnees de l'analyseur lexical

Returns

neant

5.6.2.5 initLexData()

```
TLex * initLexData (
            char * _data )
```

fonction qui reserve la memoire et initialise les donnees pour l'analyseur lexical

Parameters

in	*_data*	chaine a analyser

Returns

pointeur sur la structure de donnees creee

5.6.2.6 isSep()

```
int isSep (
            const char _symb )
```

fonction qui teste si un symbole fait partie des separateurs

Parameters

in	*_symb*	symbole a analyser

Returns

1 (vrai) si _symb est un separateur, 0 (faux) sinon

5.6.2.7 lex()

```
int lex (
            TLex * _lexData )
```

5.6.2.8 printLexData()

```
void printLexData (
            TLex * _lexData )
```

fonction qui affiche les donnees pour l'analyseur lexical

Parameters

in	*_lexData*	donnees de l'analyseur lexical

Returns

neant

5.7 tp2_lex.h File Reference

header pour analyseur lexical pour le langage JSON

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <assert.h>
```

Include dependency graph for tp2_lex.h: This graph shows which files directly or indirectly include this file:

Data Structures

- union TSymbole

  union permettant de manipuler un entier/reel/chaine pour la table des symboles
- struct TLex

  structure contenant tous les parametres/donnees pour l'analyse lexicale

Macros

- #define JSON_LEX_ERROR -1
- #define JSON_TRUE 1
- #define JSON_FALSE 2
- #define JSON_NULL 3
- #define JSON_LCB 4
- #define JSON_RCB 5
- #define JSON_LB 6
- #define JSON_RB 7
- #define JSON_COMMA 8
- #define JSON_COLON 9
- #define JSON_STRING 10
- #define JSON_INT_NUMBER 11
- #define JSON_REAL_NUMBER 12

Functions

- int isSep (const char _symb)

  fonction qui teste si un symbole fait partie des separateurs
- TLex * initLexData (char *_data)

  fonction qui reserve la memoire et initialise les donnees pour l'analyseur lexical
- void deleteLexData (TLex **_lexData)

  fonction qui supprime de la memoire les donnees pour l'analyseur lexical
- void printLexData (TLex *_lexData)

  fonction qui affiche les donnees pour l'analyseur lexical
- void addIntSymbolToLexData (TLex *_lexData, const int _val)

  fonction qui ajoute un symbole entier a la table des symboles
- void addRealSymbolToLexData (TLex *_lexData, const float _val)

  fonction qui ajoute un symbole reel a la table des symboles
- void addStringSymbolToLexData (TLex *_lexData, char *_val)

  fonction qui ajoute une chaine de caracteres a la table des symboles
- int lex (TLex *_lexData)

5.7.1 Detailed Description

header pour analyseur lexical pour le langage JSON

Author

GILBERT Dorian et DHONDT Matheo

Version

0.1

Date

11/01/2023

5.7.2 Macro Definition Documentation

5.7.2.1 JSON_COLON

```
#define JSON_COLON 9
```

entite lexicale :

5.7.2.2 JSON_COMMA

```
#define JSON_COMMA 8
```

entite lexicale ,

5.7.2.3 JSON_FALSE

```
#define JSON_FALSE 2
```

entite lexicale false

5.7.2.4 JSON_INT_NUMBER

```
#define JSON_INT_NUMBER 11
```

entite lexicale nombre entier

5.7.2.5 JSON_LB

```
#define JSON_LB 6
```

entite lexicale [

5.7.2.6 JSON_LCB

```
#define JSON_LCB 4
```

entite lexicale {

5.7.2.7 JSON_LEX_ERROR

```
#define JSON_LEX_ERROR -1
```

code d'erreur lexicale

5.7.2.8 JSON_NULL

```
#define JSON_NULL 3
```

entite lexicale null

5.7.2.9 JSON_RB

```
#define JSON_RB 7
```

entite lexicale]

5.7.2.10 JSON_RCB

```
#define JSON_RCB 5
```

entite lexicale }

5.7.2.11 JSON_REAL_NUMBER

```
#define JSON_REAL_NUMBER 12
```

entite lexicale nombre reel

5.7.2.12 JSON_STRING

```
#define JSON_STRING 10
```

entite lexicale chaine de caracteres

5.7.2.13 JSON_TRUE

```
#define JSON_TRUE 1
```

entite lexicale true

5.7.3 Function Documentation

5.7.3.1 addIntSymbolToLexData()

```
void addIntSymbolToLexData (
            TLex * _lexData,
            const int _val )
```

fonction qui ajoute un symbole entier a la table des symboles

Parameters

in,out	*_lexData*	donnees de l'analyseur lexical
in	*_val*	valeur entiere a ajouter

Returns

neant

5.7.3.2 addRealSymbolToLexData()

```
void addRealSymbolToLexData (
            TLex * _lexData,
            const float _val )
```

fonction qui ajoute un symbole reel a la table des symboles

Parameters

in,out	*_lexData*	donnees de l'analyseur lexical
in	*_val*	valeur reelle a ajouter

5.7.3.3 addStringSymbolToLexData()

```
void addStringSymbolToLexData (
            TLex * _lexData,
            char * _val )
```

fonction qui ajoute une chaine de caracteres a la table des symboles

Parameters

in,out	*_lexData*	donnees de l'analyseur lexical
in	*_val*	chaine a ajouter

5.7.3.4 deleteLexData()

```
void deleteLexData (
            TLex ** _lexData )
```

fonction qui supprime de la memoire les donnees pour l'analyseur lexical

Parameters

in,out	*_lexData*	donnees de l'analyseur lexical

Returns

neant

5.7.3.5 initLexData()

```
TLex* initLexData (
            char * _data )
```

fonction qui reserve la memoire et initialise les donnees pour l'analyseur lexical

Parameters

in	*_data*	chaine a analyser

Returns

pointeur sur la structure de donnees creee

5.7.3.6 isSep()

```
int isSep (
            const char _symb )
```

fonction qui teste si un symbole fait partie des separateurs

Parameters

in	*_symb*	symbole a analyser

Returns

1 (vrai) si _symb est un separateur, 0 (faux) sinon

5.7.3.7 lex()

```
int lex (
            TLex * _lexData )
```

5.7.3.8 printLexData()

```
void printLexData (
            TLex * _lexData )
```

fonction qui affiche les donnees pour l'analyseur lexical

Parameters

in	*_lexData*	donnees de l'analyseur lexical

Returns

neant

5.8 tp3_a.c File Reference

Analyseur syntaxique LR.

```
#include "tp3_a.h"
```
Include dependency graph for tp3_a.c:

Functions

- int analyseurLR (char *_chaine)
- int main (int argc, char *argv[ ])

5.8.1 Detailed Description

Analyseur syntaxique LR.

Author

GILBERT Dorian et DHONDT Matheo

Version

0.1

Date

20/01/2023

5.8.2 Function Documentation

5.8.2.1 analyseurLR()

```
int analyseurLR (
            char * _chaine )
```

5.8.2.2 main()

```
int main (
            int argc,
            char * argv[] )
```

5.9 tp3_a.h File Reference

Header pour analyseur syntaxique LR pour le langage JSON.

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "pile.h"
#include "tp2_lex.h"
```

Include dependency graph for tp3_a.h: This graph shows which files directly or indirectly include this file:

Functions

- int analyseurLR (char *_chaine)

Variables

- int TableAction [26][18]
- int TableGoto [26][18]
- int taillePartieDroiteRegle [16] = {2,3,1,3,3,2,3,1,3,1,1,1,1,1,1,1}
- int elementPartieGaucheRegle [16] = {13,13,14,14,15,16,16,17,17,18,18,18,18,18,18,18}

5.9.1 Detailed Description

Header pour analyseur syntaxique LR pour le langage JSON.

Author

GILBERT Dorian et DHONDT Matheo

Version

0.1

Date

26/01/2023

5.9.2 Function Documentation

5.9.2.1 analyseurLR()

```
int analyseurLR (
            char * _chaine )
```

5.9.3 Variable Documentation

5.9.3.1 elementPartieGaucheRegle

```
int elementPartieGaucheRegle[16] = {13,13,14,14,15,16,16,17,17,18,18,18,18,18,18,18}
```

Table contenant l'élément en partie gauche de chaque règle

5.9.3.2 TableAction

```
int TableAction[26][18]
```

Initial value:

```
=

{
{'d', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'a', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 'd', 'e', 'e', 'e', 'e', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e',  3 , 'e', 'e', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e',  1 , 'e',  1 ,  1 , 'e', 'e', 'e', 'e', 'e', 'e',  1 , 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 'e', 'e', 'e', 'e', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e',  2 , 'e',  2 ,  2 , 'e', 'e', 'e', 'e', 'e', 'e',  2 , 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 'e', 'e', 'e', 'e', 'e', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'d', 'e', 'd', 'e', 'e', 'e', 'd', 'd', 'd', 'd', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e',  4 , 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 12 , 'e', 12 , 12 , 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 13 , 'e', 13 , 13 , 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e',  5 , 'e', 'e',  5 , 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'d', 'e', 'd', 'd', 'e', 'e', 'd', 'd', 'd', 'd', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 10 , 'e', 10 , 10 , 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 11 , 'e', 11 , 11 , 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 14 , 'e', 14 , 14 , 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 15 , 'e', 15 , 15 , 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 16 , 'e', 16 , 16 , 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 'e', 'e', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 'e', 'e', 8  , 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e',  6 , 'e', 6  ,  6 , 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e',  7 , 'e', 7  ,  7 , 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'d', 'e', 'd', 'e', 'e', 'e', 'd', 'd', 'd', 'd', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e'},
{'e', 'e', 'e', 9  , 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e'}}
```

Table contenant les actions à effectuer (Acc='a' ; Deplacement='d' ; Reduction=<un numéro="" de="" règle="">="> ; Erreur='e')

5.9.3.3 TableGoto

```
int TableGoto[26][18]
```

Initial value:

```
=

{
{ 2,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 0,-1,-1,-1,-1,-1},
{-1, 5,-1,-1,-1,-1, 6,-1,-1,-1,-1,-1, 3, 4,-1,-1,-1},
{-1, 7,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1, 8,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1, 9,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1, 6,-1,-1,-1,-1,-1,10, 4,-1,-1,-1},
{ 2,-1,14,-1,-1,-1,15,16,17,18,19,-1,11,-1,-1,12,-1,13},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{ 2,-1,14,22,-1,-1,15,16,17,18,19,-1,11,-1,-1,12,20,21},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,23,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,24,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{ 2,-1,14,-1,-1,-1,15,16,17,18,19,-1,11,-1,-1,12,25,21},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1}}
```

Table contenant les états vers lesquels naviguer (Erreur/Pas d'état=-1)

5.9.3.4 taillePartieDroiteRegle

```
int taillePartieDroiteRegle[16] = {2,3,1,3,3,2,3,1,3,1,1,1,1,1,1,1}
```

Table contenant la taille de la partie droite de chaque règle

Index