

TP3

Generated by Doxygen 1.8.13

Contents

1	Troisième exercice du TP IL2	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Data Structure Documentation	7
4.1	_json_array Struct Reference	7
4.1.1	Field Documentation	7
4.1.1.1	elements	7
4.1.1.2	size	7
4.2	_json_object Struct Reference	8
4.2.1	Field Documentation	8
4.2.1.1	members	8
4.2.1.2	size	8
4.3	_json_pair Struct Reference	8
4.3.1	Field Documentation	8
4.3.1.1	string	9
4.3.1.2	value	9
4.4	_json_value_container Struct Reference	9
4.4.1	Field Documentation	9
4.4.1.1	type	9
4.4.1.2	value	9

4.5	JsonArray Struct Reference	10
4.5.1	Detailed Description	10
4.6	JsonObject Struct Reference	10
4.6.1	Detailed Description	10
4.7	JsonPair Struct Reference	10
4.7.1	Detailed Description	10
4.8	JsonValue Union Reference	11
4.8.1	Detailed Description	11
4.8.2	Field Documentation	11
4.8.2.1	array	11
4.8.2.2	constant	11
4.8.2.3	integer	11
4.8.2.4	object	12
4.8.2.5	real	12
4.8.2.6	string	12
4.9	JsonValueContainer Struct Reference	12
4.9.1	Detailed Description	12
4.10	TIntPile Struct Reference	12
4.10.1	Detailed Description	13
4.10.2	Field Documentation	13
4.10.2.1	data	13
4.10.2.2	indexSommet	13
4.10.2.3	size	13
4.11	TLex Struct Reference	13
4.11.1	Detailed Description	14
4.11.2	Field Documentation	14
4.11.2.1	data	14
4.11.2.2	nbLignes	14
4.11.2.3	nbSymboles	14
4.11.2.4	startPos	14

4.11.2.5	tableSymboles	14
4.11.2.6	tailleTableSymboles	14
4.12	TSymbole Union Reference	15
4.12.1	Detailed Description	15
4.12.2	Field Documentation	15
4.12.2.1	chaine	15
4.12.2.2	entier	15
4.12.2.3	reel	15
4.12.2.4	type	15
4.12.2.5	val	16
4.13	TVoidPile Struct Reference	16
4.13.1	Detailed Description	16
4.13.2	Field Documentation	16
4.13.2.1	data	16
4.13.2.2	indexSommet	16
4.13.2.3	size	16
5	File Documentation	17
5.1	json_tree.c File Reference	17
5.2	json_tree.h File Reference	17
5.2.1	Detailed Description	18
5.2.2	Macro Definition Documentation	19
5.2.2.1	CSTE_JSON_FALSE	19
5.2.2.2	CSTE_JSON_NULL	19
5.2.2.3	CSTE_JSON_TRUE	19
5.2.3	Typedef Documentation	19
5.2.3.1	JsonArray	19
5.2.3.2	JsonObject	19
5.2.3.3	JsonPair	19
5.2.3.4	JsonValueContainer	20
5.2.3.5	ValueType	20

5.2.4	Enumeration Type Documentation	20
5.2.4.1	_value_type	20
5.2.5	Function Documentation	20
5.2.5.1	CreateJsonArray()	20
5.2.5.2	CreateJsonObject()	21
5.2.5.3	CreateJsonPair()	21
5.2.5.4	CreateJsonValueContainer()	21
5.2.5.5	DeleteJsonArray()	21
5.2.5.6	DeleteJsonObject()	21
5.2.5.7	DeleteJsonPair()	22
5.2.5.8	DeleteJsonValueContainer()	22
5.2.5.9	GetJsonValueContainer()	22
5.2.5.10	InsertJsonArray()	22
5.2.5.11	InsertJsonObject()	22
5.2.5.12	PrintDotJsonArray()	22
5.2.5.13	PrintDotJsonObject()	23
5.2.5.14	PrintJsonArray()	23
5.2.5.15	PrintJsonObject()	23
5.2.5.16	PrintJsonPair()	23
5.2.5.17	PrintJsonValueContainer()	23
5.2.5.18	UpdateJsonPair()	23
5.2.5.19	UpdateJsonValueContainer()	24
5.3	pile.c File Reference	24
5.3.1	Detailed Description	25
5.3.2	Macro Definition Documentation	25
5.3.2.1	_DEFAULT_PILE_SIZE	25
5.3.3	Function Documentation	25
5.3.3.1	deleteIntPile()	25
5.3.3.2	deleteVoidPile()	25
5.3.3.3	depilerInt()	26

5.3.3.4	depilerVoid()	26
5.3.3.5	empilerInt()	27
5.3.3.6	empilerVoid()	27
5.3.3.7	initIntPile()	27
5.3.3.8	initVoidPile()	27
5.3.3.9	printIntPile()	27
5.3.3.10	printVoidPile()	28
5.3.3.11	sommetInt()	28
5.3.3.12	sommetVoid()	28
5.4	pile.h File Reference	29
5.4.1	Detailed Description	30
5.4.2	Function Documentation	30
5.4.2.1	deleteIntPile()	30
5.4.2.2	deleteVoidPile()	30
5.4.2.3	depilerInt()	31
5.4.2.4	depilerVoid()	31
5.4.2.5	empilerInt()	31
5.4.2.6	empilerVoid()	32
5.4.2.7	initIntPile()	32
5.4.2.8	initVoidPile()	32
5.4.2.9	printIntPile()	32
5.4.2.10	printVoidPile()	33
5.4.2.11	sommetInt()	33
5.4.2.12	sommetVoid()	33
5.5	README.md File Reference	34
5.6	tp2_lex.c File Reference	34
5.6.1	Detailed Description	34
5.6.2	Function Documentation	35
5.6.2.1	addIntSymbolToLexData()	35
5.6.2.2	addRealSymbolToLexData()	35

5.6.2.3	addStringSymbolToLexData()	35
5.6.2.4	deleteLexData()	36
5.6.2.5	initLexData()	36
5.6.2.6	isSep()	36
5.6.2.7	lex()	37
5.6.2.8	printLexData()	37
5.7	tp2_lex.h File Reference	37
5.7.1	Detailed Description	38
5.7.2	Macro Definition Documentation	39
5.7.2.1	JSON_COLON	39
5.7.2.2	JSON_COMMA	39
5.7.2.3	JSON_FALSE	39
5.7.2.4	JSON_INT_NUMBER	39
5.7.2.5	JSON_LB	39
5.7.2.6	JSON_LCB	39
5.7.2.7	JSON_LEX_ERROR	39
5.7.2.8	JSON_NULL	40
5.7.2.9	JSON_RB	40
5.7.2.10	JSON_RCB	40
5.7.2.11	JSON_REAL_NUMBER	40
5.7.2.12	JSON_STRING	40
5.7.2.13	JSON_TRUE	40
5.7.3	Function Documentation	40
5.7.3.1	addIntSymbolToLexData()	40
5.7.3.2	addRealSymbolToLexData()	41
5.7.3.3	addStringSymbolToLexData()	41
5.7.3.4	deleteLexData()	41
5.7.3.5	initLexData()	42
5.7.3.6	isSep()	42
5.7.3.7	lex()	42

5.7.3.8	printLexData()	43
5.8	tp3_a.c File Reference	43
5.8.1	Detailed Description	43
5.8.2	Function Documentation	43
5.8.2.1	analyseurLR()	44
5.8.2.2	main()	44
5.9	tp3_a.h File Reference	44
5.9.1	Detailed Description	44
5.9.2	Function Documentation	45
5.9.2.1	analyseurLR()	45
5.9.3	Variable Documentation	45
5.9.3.1	elementPartieGaucheRegle	45
5.9.3.2	TableAction	45
5.9.3.3	TableGoto	46
5.9.3.4	taillePartieDroiteRegle	46
	Index	47

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 transférer 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:

_json_array	7
_json_object	8
_json_pair	8
_json_value_container	9
JsonArray	
Pour les tableau "array"	10
JsonObject	
Pour stoker un objet JSON	10
JsonPair	
Pour stocker une "pair" string : value	10
JsonValue	
Pour stocker "value"	11
JsonValueContainer	
Pour stocker "value" et son type	12
TIntPile	
Structure contenant une pile d'entiers	12
TLex	
Structure contenant tous les parametres/donnees pour l'analyse lexicale	13
TSymbole	
Union permettant de manipuler un entier/reel/chaine pour la table des symboles	15
TVoidPile	
Structure contenant une pile de pointeur void *	16

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

json_tree.c	Implements the memory management of a json tree	17
json_tree.h	Implements the memory management of a json tree	17
pile.c	Management of a stack	24
pile.h	Management of a stack	29
tp2_lex.c	Lexical analyzer for the JSON language	34
tp2_lex.h	Header for lexical analyzer for the JSON language	37
tp3_a.c	Syntaxical analyzer LR	43
tp3_a.h	Header for syntaxical analyzer LR for the JSON language	44

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
```

chaîne 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 PrintJsonObject()

```
char* PrintJsonObject (
    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 default 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	<i>_pile</i>	: adresse du pointeur sur la pile a liberer
----	--------------	---

Returns

neant

5.3.3.3 depilerInt()

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

dépiler un entier

Parameters

in	<i>_pile</i>	: 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	<i>_pile</i>	: 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	<i>_pile</i>	: 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	<i>_pile</i>	: pile a afficher
----	--------------	-------------------

Returns

neant

5.3.3.11 sommetInt()

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

renvoie la valeur du sommet (sans depiler)

Parameters

in	<i>_pile</i>	: 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	<code>_pile</code>	: 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	<i>_pile</i>	: adresse du pointeur sur la pile a liberer
----	--------------	---

Returns

neant

5.4.2.3 depilerInt()

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

dépiler un entier

Parameters

in	<i>_pile</i>	: 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	<i>_pile</i>	: 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	<i>_pile</i>	: 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	<i>_pile</i>	: pile a afficher
----	--------------	-------------------

Returns

neant

5.4.2.11 sommetInt()

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

renvoie la valeur du sommet (sans depiler)

Parameters

in	<i>_pile</i>	: 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	<code>_pile</code>	: 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 Mattheo

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	<i>_lexData</i>	donnees de l'analyseur lexical
in	<i>_val</i>	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	<i>_lexData</i>	donnees de l'analyseur lexical
in	<i>_val</i>	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	<i>_lexData</i>	donnees de l'analyseur lexical
in	<i>_val</i>	chaîne 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	<i>_lexData</i>	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	<i>_data</i>	chaîne 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	<code>_symb</code>	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	<code>__lexData</code>	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 Mattheo

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	<i>_lexData</i>	donnees de l'analyseur lexical
in	<i>_val</i>	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	<i>_lexData</i>	donnees de l'analyseur lexical
in	<i>_val</i>	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	<i>_lexData</i>	donnees de l'analyseur lexical
in	<i>_val</i>	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	<i>_lexData</i>	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	<i>_data</i>	chaîne 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	<i>_symb</i>	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	<code>_lexData</code>	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}
- int [elementPartieGaucheRegle](#) [16] = {13,13,14,14,15,16,16,17,17,18,18,18,18,18,18}

5.9.1 Detailed Description

Header pour analyseur syntaxique LR pour le langage JSON.

Author

GILBERT Dorian et DHONDT Mattheo

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', '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', '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', 'e', 'd', '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', '13', 'e', '13', '13', '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', '11', 'e', '11', '11', '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', '15', 'e', '15', '15', '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', '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')

Index

- `_DEFAULT_PILE_SIZE`
 - pile.c, [25](#)
- `_json_array`, [7](#)
 - elements, [7](#)
 - size, [7](#)
- `_json_object`, [8](#)
 - members, [8](#)
 - size, [8](#)
- `_json_pair`, [8](#)
 - string, [8](#)
 - value, [9](#)
- `_json_value_container`, [9](#)
 - type, [9](#)
 - value, [9](#)
- `_value_type`
 - json_tree.h, [20](#)

- `addIntSymbolToLexData`
 - tp2_lex.c, [35](#)
 - tp2_lex.h, [40](#)
- `addRealSymbolToLexData`
 - tp2_lex.c, [35](#)
 - tp2_lex.h, [41](#)
- `addStringSymbolToLexData`
 - tp2_lex.c, [35](#)
 - tp2_lex.h, [41](#)
- `analyseurLR`
 - tp3_a.c, [43](#)
 - tp3_a.h, [45](#)
- `array`
 - JsonValue, [11](#)

- `CSTE_JSON_FALSE`
 - json_tree.h, [19](#)
- `CSTE_JSON_NULL`
 - json_tree.h, [19](#)
- `CSTE_JSON_TRUE`
 - json_tree.h, [19](#)
- `chaîne`
 - TSymbole, [15](#)
- `constant`
 - JsonValue, [11](#)
- `CreateJsonArray`
 - json_tree.h, [20](#)
- `CreateJsonObject`
 - json_tree.h, [20](#)
- `CreateJsonPair`
 - json_tree.h, [21](#)
- `CreateJsonValueContainer`
 - json_tree.h, [21](#)

- `data`
 - TIntPile, [13](#)
 - TLex, [14](#)
 - TVoidPile, [16](#)
- `deleteIntPile`
 - pile.c, [25](#)
 - pile.h, [30](#)
- `DeleteJsonArray`
 - json_tree.h, [21](#)
- `DeleteJsonObject`
 - json_tree.h, [21](#)
- `DeleteJsonPair`
 - json_tree.h, [21](#)
- `DeleteJsonValueContainer`
 - json_tree.h, [22](#)
- `deleteLexData`
 - tp2_lex.c, [36](#)
 - tp2_lex.h, [41](#)
- `deleteVoidPile`
 - pile.c, [25](#)
 - pile.h, [30](#)
- `depilerInt`
 - pile.c, [26](#)
 - pile.h, [31](#)
- `depilerVoid`
 - pile.c, [26](#)
 - pile.h, [31](#)
- `elementPartieGaucheRegle`
 - tp3_a.h, [45](#)
- `elements`
 - _json_array, [7](#)
- `empilerInt`
 - pile.c, [26](#)
 - pile.h, [31](#)
- `empilerVoid`
 - pile.c, [27](#)
 - pile.h, [32](#)
- `entier`
 - TSymbole, [15](#)
- `GetJsonValueContainer`
 - json_tree.h, [22](#)
- `indexSommet`
 - TIntPile, [13](#)
 - TVoidPile, [16](#)
- `initIntPile`
 - pile.c, [27](#)
 - pile.h, [32](#)

- initLexData
 - tp2_lex.c, 36
 - tp2_lex.h, 42
- initVoidPile
 - pile.c, 27
 - pile.h, 32
- InsertJsonArray
 - json_tree.h, 22
- InsertJsonObject
 - json_tree.h, 22
- integer
 - JsonValue, 11
- isSep
 - tp2_lex.c, 36
 - tp2_lex.h, 42
- JSON_COLON
 - tp2_lex.h, 39
- JSON_COMMA
 - tp2_lex.h, 39
- JSON_FALSE
 - tp2_lex.h, 39
- JSON_INT_NUMBER
 - tp2_lex.h, 39
- JSON_LCB
 - tp2_lex.h, 39
- JSON_LEX_ERROR
 - tp2_lex.h, 39
- JSON_LB
 - tp2_lex.h, 39
- JSON_NULL
 - tp2_lex.h, 39
- JSON_RCB
 - tp2_lex.h, 40
- JSON_REAL_NUMBER
 - tp2_lex.h, 40
- JSON_RB
 - tp2_lex.h, 40
- JSON_STRING
 - tp2_lex.h, 40
- JSON_TRUE
 - tp2_lex.h, 40
- json_tree.c, 17
- json_tree.h, 17
 - _value_type, 20
 - CSTE_JSON_FALSE, 19
 - CSTE_JSON_NULL, 19
 - CSTE_JSON_TRUE, 19
 - CreateJsonArray, 20
 - CreateJsonObject, 20
 - CreateJsonPair, 21
 - CreateJsonValueContainer, 21
 - DeleteJsonArray, 21
 - DeleteJsonObject, 21
 - DeleteJsonPair, 21
 - DeleteJsonValueContainer, 22
 - GetJsonValueContainer, 22
 - InsertJsonArray, 22
 - InsertJsonObject, 22
 - JsonArray, 19
 - JsonObject, 19
 - JsonPair, 19
 - JsonValueContainer, 19
 - PrintDotJsonArray, 22
 - PrintDotJsonObject, 22
 - PrintJsonArray, 23
 - PrintJsonObject, 23
 - PrintJsonPair, 23
 - PrintJsonValueContainer, 23
 - UpdateJsonPair, 23
 - UpdateJsonValueContainer, 23
 - ValueType, 20
- JsonArray, 10
 - json_tree.h, 19
- JsonObject, 10
 - json_tree.h, 19
- JsonPair, 10
 - json_tree.h, 19
- JsonValue, 11
 - array, 11
 - constant, 11
 - integer, 11
 - object, 11
 - real, 12
 - string, 12
- JsonValueContainer, 12
 - json_tree.h, 19
- lex
 - tp2_lex.c, 37
 - tp2_lex.h, 42
- main
 - tp3_a.c, 44
- members
 - _json_object, 8
- nbLignes
 - TLex, 14
- nbSymboles
 - TLex, 14
- object
 - JsonValue, 11
- pile.c, 24
 - _DEFAULT_PILE_SIZE, 25
 - deleteIntPile, 25
 - deleteVoidPile, 25
 - depilerInt, 26
 - depilerVoid, 26
 - empilerInt, 26
 - empilerVoid, 27
 - initIntPile, 27
 - initVoidPile, 27
 - printIntPile, 27
 - printVoidPile, 28
 - sommetInt, 28

- sommetVoid, 28
- pile.h, 29
 - deleteIntPile, 30
 - deleteVoidPile, 30
 - depilerInt, 31
 - depilerVoid, 31
 - empilerInt, 31
 - empilerVoid, 32
 - initIntPile, 32
 - initVoidPile, 32
 - printIntPile, 32
 - printVoidPile, 33
 - sommetInt, 33
 - sommetVoid, 33
- PrintDotJsonArray
 - json_tree.h, 22
- PrintDotJsonObject
 - json_tree.h, 22
- printIntPile
 - pile.c, 27
 - pile.h, 32
- PrintJsonArray
 - json_tree.h, 23
- PrintJsonObject
 - json_tree.h, 23
- PrintJsonPair
 - json_tree.h, 23
- PrintJsonValueContainer
 - json_tree.h, 23
- printLexData
 - tp2_lex.c, 37
 - tp2_lex.h, 42
- printVoidPile
 - pile.c, 28
 - pile.h, 33
- README.md, 34
- real
 - JsonValue, 12
- reel
 - TSymbole, 15
- size
 - _json_array, 7
 - _json_object, 8
 - TIntPile, 13
 - TVoidPile, 16
- sommetInt
 - pile.c, 28
 - pile.h, 33
- sommetVoid
 - pile.c, 28
 - pile.h, 33
- startPos
 - TLex, 14
- string
 - _json_pair, 8
 - JsonValue, 12
- TIntPile, 12
 - data, 13
 - indexSommet, 13
 - size, 13
- TLex, 13
 - data, 14
 - nbLignes, 14
 - nbSymboles, 14
 - startPos, 14
 - tableSymboles, 14
 - tailleTableSymboles, 14
- TSymbole, 15
 - chaine, 15
 - entier, 15
 - reel, 15
 - type, 15
 - val, 15
- TVoidPile, 16
 - data, 16
 - indexSommet, 16
 - size, 16
- TableAction
 - tp3_a.h, 45
- TableGoto
 - tp3_a.h, 45
- tableSymboles
 - TLex, 14
- taillePartieDroiteRegle
 - tp3_a.h, 46
- tailleTableSymboles
 - TLex, 14
- tp2_lex.c, 34
 - addIntSymbolToLexData, 35
 - addRealSymbolToLexData, 35
 - addStringSymbolToLexData, 35
 - deleteLexData, 36
 - initLexData, 36
 - isSep, 36
 - lex, 37
 - printLexData, 37
- tp2_lex.h, 37
 - addIntSymbolToLexData, 40
 - addRealSymbolToLexData, 41
 - addStringSymbolToLexData, 41
 - deleteLexData, 41
 - initLexData, 42
 - isSep, 42
 - JSON_COLON, 39
 - JSON_COMMA, 39
 - JSON_FALSE, 39
 - JSON_INT_NUMBER, 39
 - JSON_LCB, 39
 - JSON_LEX_ERROR, 39
 - JSON_LB, 39
 - JSON_NULL, 39
 - JSON_RCB, 40
 - JSON_REAL_NUMBER, 40
 - JSON_RB, 40

- JSON_STRING, [40](#)
- JSON_TRUE, [40](#)
- lex, [42](#)
- printLexData, [42](#)
- tp3_a.c, [43](#)
 - analyseurLR, [43](#)
 - main, [44](#)
- tp3_a.h, [44](#)
 - analyseurLR, [45](#)
 - elementPartieGaucheRegle, [45](#)
 - TableAction, [45](#)
 - TableGoto, [45](#)
 - taillePartieDroiteRegle, [46](#)
- type
 - _json_value_container, [9](#)
 - TSymbole, [15](#)
- UpdateJsonPair
 - json_tree.h, [23](#)
- UpdateJsonValueContainer
 - json_tree.h, [23](#)
- val
 - TSymbole, [15](#)
- value
 - _json_pair, [9](#)
 - _json_value_container, [9](#)
- ValueType
 - json_tree.h, [20](#)