A simple tiger compiler

Generated by Doxygen 1.9.8

Chapter 1

Topic Index

1.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 2

Struct Index

2.1 Struct List

Here are the structs with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Topic Documentation

4.1 SLP_interpreter

an interpreter to Straight-Line Program(SLP) language

Files

- file grammar_interpreter.hpp
  realize of grammar

4.1.1 Detailed Description

an interpreter to Straight-Line Program(SLP) language

4.1.2 Function Documentation

4.1.2.1 max()

```
template<typename T = int>
T max (
            T a,
            T b )
```

4.1.3 <tt>__main__</tt> file

4.2 Utils

realize of utils.hpp

realize of utils.hpp

USE ASSERT!!!

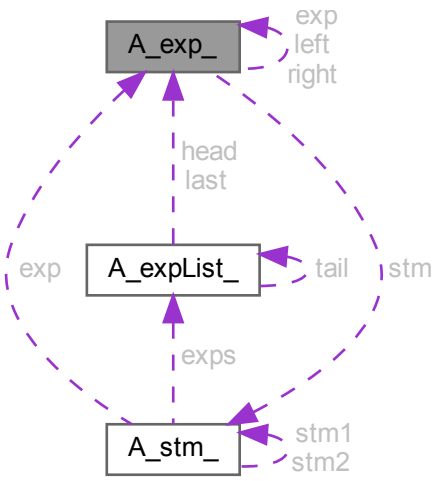
Chapter 5

Struct Documentation

5.1 A_exp_ Struct Reference

`#include <grammar_interpreter.hpp>`

Collaboration diagram for A_exp_:



Public Types

Data Fields

5.1.1 Detailed Description

```
Exp -> id
Exp -> num
Exp -> Exp Binop Exp
Exp -> (Stm, Exp)
```

5.1.2 Member Enumeration Documentation

5.1.2.1 anonymous enum

```
anonymous enum
```

Enumerator

A_idExp	
A_numExp	
A_opExp	
A_eseqExp	

5.1.3 Field Documentation

5.1.3.1 [struct]

```
struct { ... } eseq
```

5.1.3.2 exp

```
A_exp exp
```

5.1.3.3 id

```
string id
```

5.1.3.4 []

```
enum { ... } kind
```

5.1.3.5 left

```
A_exp left
```

5.1.3.6 num

```
int num
```

5.1.3.7 [struct]

```
struct { ... } op
```

5.1.3.8 oper

```
A_binop oper
```

5.1.3.9 right

```
A_exp right
```

5.1.3.10 stm

```
A_stm stm
```

5.1.3.11 [union]

```
union { ... } u
```

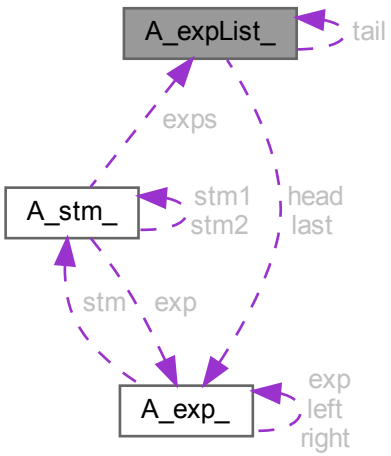
The documentation for this struct was generated from the following file:

- src/SLP_interpreter/grammar_interpreter.hpp

5.2 A_expList_ Struct Reference

```
#include <grammar_interpreter.hpp>
```

Collaboration diagram for A_expList_:



Public Types

Data Fields

5.2.1 Detailed Description

```
ExpList -> Exp, ExpList
ExpList -> Exp
```

5.2.2 Member Enumeration Documentation

5.2.2.1 anonymous enum

```
anonymous enum
```

Enumerator

A_pairExpList	
A_lastExpList	

5.2.3 Field Documentation

5.2.3.1 head

```
A_exp head
```

5.2.3.2 []

```
enum { ...  } kind
```

5.2.3.3 last

```
A_exp last
```

5.2.3.4 [struct]

```
struct { ...  } pair
```

5.2.3.5 tail

```
A_expList tail
```

5.2.3.6 [union]

```
union { ... } u
```

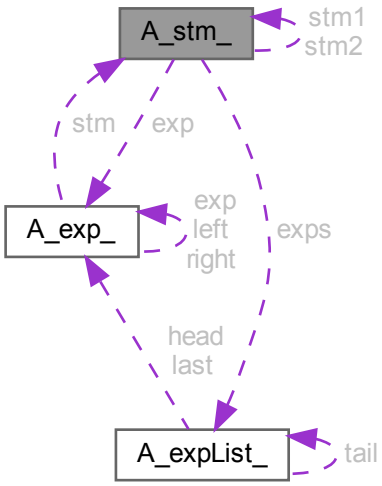
The documentation for this struct was generated from the following file:

- src/SLP_interpreter/grammar_interpreter.hpp

5.3 A_stm_ Struct Reference

```
#include <grammar_interpreter.hpp>
```

Collaboration diagram for A_stm_:



Public Types

Data Fields

5.3.1 Detailed Description

```
Stm -> Stm; Stm
Stm -> id := Stm
Stm -> print(ExpList)
```

5.3.2 Member Enumeration Documentation

5.3.2.1 anonymous enum

```
anonymous enum
```

Enumerator

A_compoundStm	
A_assignStm	
A_printStm	

5.3.3 Field Documentation

5.3.3.1 [struct]

```
struct { ... } assign
```

5.3.3.2 [struct]

```
struct { ... } compound
```

5.3.3.3 exp

```
A_exp exp
```

5.3.3.4 exps

```
A_expList exps
```

5.3.3.5 id

```
string id
```

5.3.3.6 []

```
enum { ... } kind
```

5.3.3.7 [struct]

```
struct { ... } print
```

5.3.3.8 stm1

```
A_stm stm1
```

5.3.3.9 stm2

```
A_stm stm2
```

5.3.3.10 [union]

```
union { ... } u
```

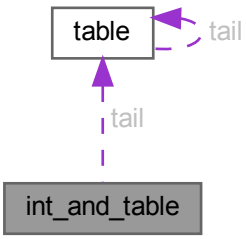
The documentation for this struct was generated from the following file:

- src/SLP_interpreter/grammar_interpreter.hpp

5.4 int_and_table Struct Reference

```
#include <main.hpp>
```

Collaboration diagram for int_and_table:



Data Fields

5.4.1 Field Documentation

5.4.1.1 i

```
int i
```

5.4.1.2 tail

```
Table_ tail
```

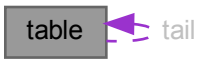
The documentation for this struct was generated from the following file:

- src/SLP_interpreter/main.hpp

5.5 table Struct Reference

`#include <main.hpp>`

Collaboration diagram for table:



Data Fields

5.5.1 Field Documentation

5.5.1.1 id

`string id`

5.5.1.2 tail

`Table_ tail`

5.5.1.3 value

`int value`

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

- src/SLP_interpreter/main.hpp

Chapter 6

File Documentation

6.1 src/SLP_interpreter/grammar_interpreter.cpp File Reference

6.1.1 Function Documentation

6.1.1.1 A_AssignStm()

```
A_stm A_AssignStm (
            string id,
            A_exp exp )
```

6.1.1.2 A_CompoundStm()

```
A_stm A_CompoundStm (
            A_stm stm1,
            A_stm stm2 )
```

6.1.1.3 A_EseqExp()

```
A_exp A_EseqExp (
            A_stm stm,
            A_exp exp )
```

6.1.1.4 A_IdExp()

```
A_exp A_IdExp (
            string id )
```

6.1.1.5 A_LastExpList()

```
A_expList A_LastExpList (
            A_exp last )
```

6.1.1.6 A_NumExp()

```
A_exp A_NumExp (
            int num )
```

6.1.1.7 A_OpExp()

```
A_exp A_OpExp (
            A_exp left,
            A_binop oper,
            A_exp right )
```

6.1.1.8 A_PairExpList()

```
A_expList A_PairExpList (
            A_exp head,
            A_expList tail )
```

6.1.1.9 A_PrintStm()

```
A_stm A_PrintStm (
            A_expList exps )
```

6.2 src/SLP_interpreter/grammar_interpreter.hpp File Reference

realize of grammar

6.2.1 Detailed Description

realize of grammar

Grammar atoms: stm, exp, expList, id, num, ...
Grammar rules: A_compoundStm, A_assignStm, A_printStm, ...

```
* For example, grammar atoms ga have rules gr1, gr2, ...
*  using A_ga = ptr to struct A_ga_
*  struct A_ga_:
*      field enum   ->  rules tokens
*      field union  ->  rules components
* constructo of rule:
*  auto A_gr(A_ga components ...) -> pointer to struct A_ga;
*
```

6.2.2 Typedef Documentation

6.2.2.1 A_exp

```
using A_exp = A_exp_ *
```

6.2.2.2 A_expList

```
using A_expList = A_expList_ *
```

6.2.2.3 A_stm

```
using A_stm = A_stm_ *
```

grammars was defined to type
for with data grammars, we use pointer to struct
for without data grammars, we use enum

6.2.3 Enumeration Type Documentation

6.2.3.1 A_binop

```
enum A_binop
```

Enumerator

A_plus	
A_minus	
A_times	
A_div	

6.2.4 Function Documentation

6.2.4.1 A_AssignStm()

```
A_stm A_AssignStm (
            string id,
            A_exp exp )
```

6.2.4.2 A_CompoundStm()

```
A_stm A_CompoundStm (
            A_stm stm1,
            A_stm stm2 )
```

6.2.4.3 A_EseqExp()

```
A_exp A_EseqExp (
            A_stm stm,
            A_exp exp )
```

6.2.4.4 A_IdExp()

```
A_exp A_IdExp (
            string id )
```

6.2.4.5 A_LastExpList()

```
A_expList A_LastExpList (
            A_exp last )
```

6.2.4.6 A_NumExp()

```
A_exp A_NumExp (
            int num )
```

6.2.4.7 A_OpExp()

```
A_exp A_OpExp (
            A_exp left,
            A_binop oper,
            A_exp right )
```

6.2.4.8 A_PairExpList()

```
A_expList A_PairExpList (
            A_exp head,
            A_expList tail )
```

6.2.4.9 A_PrintStm()

```
A_stm A_PrintStm (
            A_expList exps )
```

6.3 grammar_interpreter.hpp

Go to the documentation of this file.

```
00001
00026 #include "../utils.hpp"
00027
00028 using A_stm = struct A_stm_ *;
00029 using A_exp = struct A_exp_ *;
00030 using A_expList = struct A_expList_ *;
00031 enum A_binop
00032 {
00033     A_plus,  // +
00034     A_minus, // -
00035     A_times, // *
00036     A_div    // /
00037 };
00038
00048 struct A_stm_
00049 {
```

```
00050     enum
00051     {
00052         A_compoundStm, // Stm -> Stm; Stm
00053         A_assignStm,   // Stm -> id := Exp
00054         A_printStm     // Stm -> print(ExpList)
00055     } kind;
00056     union
00057     {
00058         struct
00059         {
00060             A_stm stm1, stm2;
00061         } compound;
00062         struct
00063         {
00064             string id;
00065             A_exp exp;
00066         } assign;
00067         struct
00068         {
00069             A_expList exps;
00070         } print;
00071     } u;
00072 };
00073
00074 A_stm A_CompoundStm(A_stm stm1, A_stm stm2);
00075 A_stm A_AssignStm(string id, A_exp exp);
00076 A_stm A_PrintStm(A_expList exps);
00077
00088 struct A_exp_
00089 {
00090     enum
00091     {
00092         A_idExp,  // Exp -> id
00093         A_numExp, // Exp -> num
00094         A_opExp,  // Exp -> Exp Binop Exp
00095         A_eseqExp // Exp -> (Stm, Exp)
00096     } kind;
00097     union
00098     {
00099         string id;
00100         int num;
00101         struct
00102         {
00103             A_exp left;
00104             A_binop oper;
00105             A_exp right;
00106         } op;
00107         struct
00108         {
00109             A_stm stm;
00110             A_exp exp;
00111         } eseq;
00112     } u;
00113 };
00114
00115 A_exp A_IdExp(string id);
00116 A_exp A_NumExp(int num);
00117 A_exp A_OpExp(A_exp left, A_binop oper, A_exp right);
00118 A_exp A_EseqExp(A_stm stm, A_exp exp);
00119
00128 struct A_expList_
00129 {
00130     enum
00131     {
00132         A_pairExpList, // ExpList -> Exp, ExpList
00133         A_lastExpList  // ExpList -> Exp
00134     } kind;
00135     union
00136     {
00137         struct
00138         {
00139             A_exp head;
00140             A_expList tail;
00141         } pair;
00142         A_exp last;
00143     } u;
00144 };
00145
00146 A_expList A_PairExpList(A_exp head, A_expList tail);
00147 A_expList A_LastExpList(A_exp last);
```

6.4 src/SLP_interpreter/main.cpp File Reference

6.4.1 Function Documentation

6.4.1.1 interp()

```
void interp (
            A_stm s )
```

the final SLP interpreter.

6.4.1.2 interpExp()

```
IntAndTable_ interpExp (
            A_exp e,
            Table_ t )
```

interprete a exp e, and table t is the current state.

6.4.1.3 interpExpList()

```
IntAndTable_ interpExpList (
            A_expList el,
            Table_ t )
```

interprete a explist el, and table t is the current state.

6.4.1.4 interpStm()

```
Table_ interpStm (
            A_stm s,
            Table_ t )
```

interprete a stm s, and table t is the current state.

6.4.1.5 lookup()

```
int lookup (
            Table_ t,
            string key )
```

find the identifier key's value in table t

6.4.1.6 main()

```
int main ( )
```

6.4.1.7 maxargs()

```
int maxargs (
             A_stm s )
```

Count `print` args number max.

6.4.1.8 min()

```
template<typename T = int>
T min (
             T a,
             T b )
```

6.4.1.9 prog_generator()

```
A_stm prog_generator (
             void  )
```

Example Tiger Code 1.

6.4.1.10 of SLP interpreter

```
a := 5+3; b := (print(a, a-1), 10*a);
print(b); print(c1);
```

6.4.1.11 prog_generator2()

```
A_stm prog_generator2 (
             void  )
```

Example Tiger Code 2.

```
a := 5+3; b := (print(a, a-1), 10*a); print(b);
print(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10);
```

6.4.1.12 update()

```
Table_ update (
             Table_ t1,
             string id2,
             int v2 )
```

update table t1 with new identifier id2 and value t2

6.4.1.13 update_e()

```
void update_e (
             A_exp e,
             int & count,
             int & history_max,
             bool & is_in_print,
             bool & is_out_print )
```

update `count` counter and `history_max` record in Exp

6.4.1.14 update_el()

```
void update_el (
            A_expList el,
            int & count,
            int & history_max,
            bool & is_in_print,
            bool & is_out_print )
```

update `count` counter and `history_max` record in ExpList

6.4.1.15 update_s()

```
void update_s (
            A_stm s,
            int & count,
            int & history_max,
            bool & is_in_print,
            bool & is_out_print )
```

update `count` counter and `history_max` record in Stm

6.5 src/SLP_interpreter/main.hpp File Reference

Data Structures

- struct int_and_table

6.5.1 Typedef Documentation

6.5.1.1 IntAndTable_

```
using IntAndTable_ = int_and_table *
```

6.5.1.2 Table_

```
using Table_ = table *
```

`table` and `int_and_table` are used to store identifiers.
A `table` is used to store a stm and it's value.
A `int_and_table` is combines a table and a value, which is the expression's return value.
In fact, these tables are linked note list.

6.5.2 Function Documentation

6.5.2.1 interp()

```
void interp (
            A_stm s )
```

the final SLP interpreter.

6.5.2.2 interpExp()

```
IntAndTable_ interpExp (
             A_exp e,
             Table_ t )
```

interprete a exp e, and table t is the current state.

6.5.2.3 interpExpList()

```
IntAndTable_ interpExpList (
             A_exp e,
             Table_ t )
```

6.5.2.4 interpStm()

```
Table_ interpStm (
             A_stm s,
             Table_ t )
```

interprete a stm s, and table t is the current state.

6.5.2.5 maxargs()

```
int maxargs (
             A_stm s )
```

Count `print` args number max.

6.5.2.6 Table()

```
Table_ Table (
             string id,
             int value,
             struct table * tail )
```

6.5.2.7 update_e()

```
void update_e (
             A_exp e,
             int & count,
             int & history_max,
             bool & is_in_print,
             bool & is_out_print )
```

update `count` counter and `history_max` record in Exp

6.5.2.8 update_el()

```
void update_el (
            A_expList el,
            int & count,
            int & history_max,
            bool & is_in_print,
            bool & is_out_print )
```

update `count` counter and `history_max` record in ExpList

6.5.2.9 update_s()

```
void update_s (
            A_stm s,
            int & count,
            int & history_max,
            bool & is_in_print,
            bool & is_out_print )
```

update `count` counter and `history_max` record in Stm

6.6 main.hpp

Go to the documentation of this file.

```
00001
00006 #include "grammar_interpreter.hpp"
00007
00008 void update_e(A_exp e, int &count, int &history_max, bool &is_in_print, bool &is_out_print);
00009 void update_el(A_expList el, int &count, int &history_max, bool &is_in_print, bool &is_out_print);
00010 void update_s(A_stm s, int &count, int &history_max, bool &is_in_print, bool &is_out_print);
00011 int maxargs(A_stm);
00012
00013 void interp(A_stm);
00014
00024 using Table_ = struct table *;
00025 using IntAndTable_ = struct int_and_table *;
00026
00027 struct table {
00028     string id;
00029     int value;
00030     Table_ tail;
00031 };
00032
00033 Table_ Table(string id, int value, struct table *tail) {
00034     Table_ t = (Table_)checked_malloc(sizeof(*t));
00035     *t = {.id = id, .value = value, .tail = tail};
00036     return t;
00037 }
00038
00039 struct int_and_table {
00040     int i;
00041     Table_ tail;
00042 };
00043
00044 Table_ interpStm(A_stm s, Table_ t);
00045 IntAndTable_ interpExp(A_exp e, Table_ t);
00046 IntAndTable_ interpExpList(A_exp e, Table_ t);
00047 void interp(A_stm);
```

6.7 src/utils.cpp File Reference

6.7.1 Function Documentation

6.7.1.1 checked_malloc()

```
void* checked_malloc (
            std::size_t size )
```

Don't use `new/delete`(C++), `malloc/calloc/free`(C)

In C++, memory allocator in <cstdlib>

Warning

malloc will return NULL (C) or nullptr (C++>=11)

6.7.1.2 String()

```
string String (
            string origin )
```

string constructor

6.8 src/utils.hpp File Reference

6.8.1 Typedef Documentation

6.8.1.1 string

```
using string = const char *
```

Warning

we will store it in heap

6.8.2 Function Documentation

6.8.2.1 checked_malloc()

```
void* checked_malloc (
            std::size_t size )
```

Don't use `new/delete`(C++), `malloc/calloc/free`(C)

noticed that, we will allocate memory for struct with union member because of the uncertain size of union, the default constructor is deleted, so new cannot used to create an struct object which with union member.

for some reason, we do not use `new/delete` in all project code, and we will use checked_malloc forever instead of `malloc/calloc`, and we never use free because of garbage collector. (Reason can be found in textbook.)

Parameters

std↩ ::size_t	the size will allocated

Returns

a `void *` pointer point to memory which is allocated by this call.

In C++, memory allocator in <cstdlib>

Warning

malloc will return NULL (C) or nullptr (C++>=11)

6.8.2.2 String()

```
string String (
             string origin )
```

string constructor

6.9 utils.hpp

Go to the documentation of this file.

```
00001
00002 #include <assert.h>
00003 #include <cstdlib>
00004 #include <string>
00005
00007 using string = const char *;
00009 string String(string);
00010
00024 void *checked_malloc(std::size_t);
```

Index