

Pico
1.0

Gerado por Doxygen 1.6.1

Mon Apr 12 21:14:31 2010

Sumário

1	Índice das Estruturas de Dados	1
1.1	Estruturas de Dados	1
2	Índice dos Arquivos	3
2.1	Lista de Arquivos	3
3	Estruturas	5
3.1	Referência da Estrutura <code>_node</code>	5
3.1.1	Descrição Detalhada	5
3.1.2	Campos e Atributos	5
3.1.2.1	<code>attribute</code>	5
3.1.2.2	<code>child</code>	5
3.1.2.3	<code>id</code>	5
3.1.2.4	<code>lexeme</code>	6
3.1.2.5	<code>num_children</code>	6
3.1.2.6	<code>num_line</code>	6
3.1.2.7	<code>type</code>	6
3.2	Referência da Estrutura <code>entry_t</code>	7
3.2.1	Descrição Detalhada	7
3.2.2	Campos e Atributos	7
3.2.2.1	<code>desloc</code>	7
3.2.2.2	<code>extra</code>	7
3.2.2.3	<code>name</code>	7
3.2.2.4	<code>size</code>	7
3.2.2.5	<code>type</code>	7
3.3	Referência da Estrutura <code>hash_entry_t</code>	8
3.3.1	Descrição Detalhada	8
3.3.2	Campos e Atributos	8
3.3.2.1	<code>dir</code>	8

3.3.2.2	entry	8
3.3.2.3	esq	8
3.3.2.4	fatorB	8
3.4	Referência da Estrutura node_tac	9
3.4.1	Descrição Detalhada	9
3.4.2	Campos e Atributos	9
3.4.2.1	inst	9
3.4.2.2	next	9
3.4.2.3	prev	9
3.5	Referência da Estrutura s_symbol_t	10
3.5.1	Descrição Detalhada	10
3.5.2	Campos e Atributos	10
3.5.2.1	entry	10
3.6	Referência da Estrutura tac	11
3.6.1	Descrição Detalhada	11
3.6.2	Campos e Atributos	11
3.6.2.1	arg1	11
3.6.2.2	arg2	11
3.6.2.3	op	11
3.6.2.4	res	11
4	Arquivos	13
4.1	Referência do Arquivo avl.h	13
4.1.1	Descrição Detalhada	14
4.1.2	Definições dos tipos	14
4.1.2.1	pNodoA	14
4.1.3	Funções	14
4.1.3.1	altura	14
4.1.3.2	balanceamentoDir	15
4.1.3.3	balanceamentoEsq	15
4.1.3.4	calculaFatorB	15
4.1.3.5	destroiAVL	15
4.1.3.6	findNodo	15
4.1.3.7	imprimeAVL	16
4.1.3.8	inicializaAVL	16
4.1.3.9	insereAVL	16
4.1.3.10	NodoVazio	16

4.1.3.11	printNiveis	16
4.1.3.12	rotDireita	17
4.1.3.13	rotDupDireita	17
4.1.3.14	rotDupEsquerda	17
4.1.3.15	rotEsquerda	17
4.2	Referência do Arquivo entry.h	18
4.2.1	Descrição Detalhada	18
4.3	Referência do Arquivo lista.h	19
4.3.1	Descrição Detalhada	19
4.3.2	Funções	19
4.3.2.1	append_inst_tac	19
4.3.2.2	cat_tac	19
4.3.2.3	create_inst_tac	20
4.3.2.4	print_inst_tac	20
4.3.2.5	print_tac	20
4.4	Referência do Arquivo node.h	21
4.4.1	Descrição Detalhada	22
4.4.2	Definições e macros	22
4.4.2.1	MAX_CHILDREN_NUMBER	22
4.4.2.2	program_node	22
4.4.3	Definições dos tipos	22
4.4.3.1	Node	22
4.4.3.2	Node_type	22
4.4.4	Funções	22
4.4.4.1	child	22
4.4.4.2	create_leaf	23
4.4.4.3	create_node	23
4.4.4.4	deep_free_node	23
4.4.4.5	height	24
4.4.4.6	nb_of_children	24
4.4.4.7	pack_nodes	24
4.5	Referência do Arquivo symbol_table.h	25
4.5.1	Descrição Detalhada	25
4.5.2	Definições e macros	26
4.5.2.1	TABLE_SIZE	26
4.5.3	Definições dos tipos	26

4.5.3.1	symbol_t	26
4.5.4	Funções	26
4.5.4.1	free_table	26
4.5.4.2	hash	26
4.5.4.3	init_table	26
4.5.4.4	insert	27
4.5.4.5	lookup	27
4.5.4.6	print_file_table	27
4.5.4.7	print_table	27

Capítulo 1

Índice das Estruturas de Dados

1.1 Estruturas de Dados

Aqui estão as estruturas de dados e suas respectivas descrições:

<code>_node</code>	5
<code>entry_t</code>	7
<code>hash_entry_t</code>	8
<code>node_tac</code>	9
<code>s_symbol_t</code> (Encapsulacao de um tipo abstrato que se chamara 'symbol_t')	10
<code>tac</code>	11

Capítulo 2

Índice dos Arquivos

2.1 Lista de Arquivos

Esta é a lista de todos os arquivos documentados e suas respectivas descrições:

avl.h	13
entry.h	18
lista.h	19
node.h	21
symbol_table.h	25

Capítulo 3

Estruturas

3.1 Referência da Estrutura `_node`

```
#include <node.h>
```

Campos de Dados

- int `num_line`
- int `id`
- char * `lexeme`
- `Node_type` `type`
- void * `attribute`
- int `num_children`
- struct `_node` * `child` [MAX_CHILDREN_NUMBER]

3.1.1 Descrição Detalhada

Estrutura de dados parcial para o nó da árvore. Trata-se de uma árvore generalizada: qualquer nó pode ter de 0 até MAX_CHILDREN_NUMBER filhos.

3.1.2 Campos e Atributos

3.1.2.1 `void* _node::attribute`

Qualquer coisa por enquanto.

3.1.2.2 `struct _node* _node::child[MAX_CHILDREN_NUMBER]` [read]

Vetor de ponteiros para os filhos dos nós.

3.1.2.3 `int _node::id`

Rótulo do nó. Cada nó deve ter um 'id' distinto.

3.1.2.4 char* _node::lexeme

irrelevante por enquanto.

3.1.2.5 int _node::num_children

Numero de filhos do nodo.

3.1.2.6 int _node::num_line

numero de linha (irrelevante por enquanto).

3.1.2.7 Node_type _node::type

Um dos valores definidos acima pelos # defines.

A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- [node.h](#)

3.2 Referência da Estrutura entry_t

```
#include <entry.h>
```

Campos de Dados

- char * [name](#)
- int [type](#)
- int [size](#)
- int [desloc](#)
- void * [extra](#)

3.2.1 Descrição Detalhada

Tipo abstrato das entradas na tabela de Hash. (Obs.: futuramente, os campos dessa struct poderao vir a ser alterados em funcao das necessidades.) Na Etapa 2, nao e necessario entender o conteudo desses campos. Sempre vao ser inseridos na tabela, e recuperado dela, ponteiros sobre tais estruturas de dados abstratas.

3.2.2 Campos e Atributos

3.2.2.1 int entry_t::desloc

Endereco da proxima variavel.

3.2.2.2 void* entry_t::extra

qualquer informacao extra.

3.2.2.3 char* entry_t::name

um string que representa o nome de uma variavel.

3.2.2.4 int entry_t::size

numero de Bytes necessarios para armazenamento.

3.2.2.5 int entry_t::type

representacao do tipo da variavel.

A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- [entry.h](#)

3.3 Referência da Estrutura `hash_entry_t`

```
#include <avl.h>
```

Campos de Dados

- struct `entry_t` * `entry`
- struct `hash_entry_t` * `dir`
- struct `hash_entry_t` * `esq`
- int `fatorB`

3.3.1 Descrição Detalhada

Entrada na tabela de hash Trata-se de uma AVL

3.3.2 Campos e Atributos

3.3.2.1 `struct hash_entry_t * hash_entry_t::dir` [`read`]

filho à direita do nó

3.3.2.2 `struct entry_t * hash_entry_t::entry` [`read`]

conteúdo do nó

3.3.2.3 `struct hash_entry_t * hash_entry_t::esq` [`read`]

filho à esquerda do nó

3.3.2.4 `int hash_entry_t::fatorB`

fator de balanceamento do nó

A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- [avl.h](#)

3.4 Referência da Estrutura node_tac

```
#include <lista.h>
```

Campos de Dados

- struct [tac](#) * [inst](#)
- struct [node_tac](#) * [next](#)
- struct [node_tac](#) * [prev](#)

3.4.1 Descrição Detalhada

Um elemento basico da lista. O campo 'inst' aponta para a informacao a ser armazenada em um elemento da lista. O resto serve para implementar a lista e seus metodos.

3.4.2 Campos e Atributos

3.4.2.1 struct [tac](#)* [node_tac::inst](#) [[read](#)]

informação a ser armazenada

3.4.2.2 struct [node_tac](#)* [node_tac::next](#) [[read](#)]

nó anterior

3.4.2.3 struct [node_tac](#)* [node_tac::prev](#) [[read](#)]

próximo nó

A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- [lista.h](#)

3.5 Referência da Estrutura `s_symbol_t`

Encapsulacao de um tipo abstrato que se chamara `'symbol_t'`.

```
#include <symbol_table.h>
```

Campos de Dados

- `hash_entry_t * entry` [TABLE_SIZE]

3.5.1 Descrição Detalhada

Encapsulacao de um tipo abstrato que se chamara `'symbol_t'`. Voce deve inserir, entre o `'typedef'` e o `'symbol_t'`, a estrutura de dados abstrata que voce ira implementar.

3.5.2 Campos e Atributos

3.5.2.1 `hash_entry_t * s_symbol_t::entry`[TABLE_SIZE]

implementação da `symbol_table`

A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- [symbol_table.h](#)

3.6 Referência da Estrutura tac

```
#include <lista.h>
```

Campos de Dados

- char * [op](#)
- char * [res](#)
- char * [arg1](#)
- char * [arg2](#)

3.6.1 Descrição Detalhada

Estrutura de dados (que implementa uma intrucao TAC, ver Etapa 4), a ser encadeada na lista. Por enquanto, o que representa é irrelevante.

3.6.2 Campos e Atributos

3.6.2.1 char* tac::arg1

"TMP0"

3.6.2.2 char* tac::arg2

"TMP1"

3.6.2.3 char* tac::op

"+", "-", ":", "if", etc...

3.6.2.4 char* tac::res

"TMP100"

A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- [lista.h](#)

Capítulo 4

Arquivos

4.1 Referência do Arquivo avl.h

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "entry.h"
```

Estruturas de Dados

- struct `hash_entry_t`

Definições de Tipos

- typedef struct `hash_entry_t` * `pNodoA`

Funções

- `pNodoA inicializaAVL ()`
Inicializar a arvore AVL.
- void `destroiAVL (pNodoA arv)`
Destroi a arvore, desalocando memoria ocupada por ela.
- `pNodoA insereAVL (pNodoA arv, entry_t *symbol)`
Insere elemento na arvore.
- `pNodoA findNodo (pNodoA arv, char *name, int *find)`
Encontra um elemento na arvore.
- void `imprimeAVL (FILE *out, pNodoA arv, int *n_nodo)`

Imprime a arvore em ordem crescente de ordenamento.

- void `printNiveis` (`pNodoA` arv, int count)
Imprime a arvore com saida formatada em camadas.
- `pNodoA` `rotEsquerda` (`pNodoA` arv)
Realizada rotacao a esquerda na arvore.
- `pNodoA` `rotDireita` (`pNodoA` arv)
Realizada rotacao a direita na arvore.
- `pNodoA` `rotDupEsquerda` (`pNodoA` arv)
Realizada rotacao dupla a esquerda na arvore.
- `pNodoA` `rotDupDireita` (`pNodoA` arv)
Realizada rotacao dupla a direita na arvore.
- `pNodoA` `balanceamentoEsq` (`pNodoA` arv)
Realizada balanceamento a esquerda na arvore.
- `pNodoA` `balanceamentoDir` (`pNodoA` arv)
Realizada balanceamento a direita na arvore.
- int `NodoVazio` (`pNodoA` arv)
Testa de o nodo é vazio.
- int `altura` (`pNodoA` arv)
Calcula a altura da arvore.
- int `calculaFatorB` (`pNodoA` arv)
Calcula o fator de balanceamento da arvore.

4.1.1 Descrição Detalhada

Versão:

1.1

4.1.2 Definições dos tipos

4.1.2.1 `typedef struct hash_entry_t * pNodoA`

ponteiro para um nó

4.1.3 Funções

4.1.3.1 `int altura (pNodoA arv)`

Calcula a altura da arvore.

Parâmetros:

arv raiz da árvore

4.1.3.2 pNodoA balanceamentoDir (pNodoA arv)

Realizada balanceamento a direita na árvore.

Parâmetros:

arv raiz da árvore.

4.1.3.3 pNodoA balanceamentoEsq (pNodoA arv)

Realizada balanceamento a esquerda na árvore.

Parâmetros:

arv raiz da árvore.

4.1.3.4 int calculaFatorB (pNodoA arv)

Calcula o fator de balanceamento da árvore.

Parâmetros:

arv raiz da árvore

4.1.3.5 void destroiAVL (pNodoA arv)

Destroi a árvore, desalocando memória ocupada por ela.

Parâmetros:

arv raiz da árvore a ser destruída.

4.1.3.6 pNodoA findNodo (pNodoA arv, char * name, int * find)

Encontra um elemento na árvore.

Parâmetros:

arv raiz da árvore.

name nome do elemento a ser pesquisado.

find endereço de um inteiro que armazena se o elemento foi encontrado (1 encontrado, 0 não).

Retorna:

ponteiro para o nó da árvore que contém o elemento pesquisado.

4.1.3.7 void imprimeAVL (FILE * *out*, pNodoA *arv*, int * *n_nodo*)

Imprime a arvore em ordem crescente de ordenamento.

Parâmetros:

out saída para imprimir a arvore.

arv raiz da arvore.

n_nodo endereco de um contador do numero de nodos.

4.1.3.8 pNodoA inicializaAVL ()

Inicializar a arvore AVL.

Retorna:

ponteiro para a raiz da arvore.

4.1.3.9 pNodoA insereAVL (pNodoA *arv*, entry_t * *symbol*)

Inserve elemento na arvore.

Parâmetros:

arv raiz da arvore.

symbol entrada a ser inserida.

Retorna:

ponteiro para a raiz da arvore.

4.1.3.10 int NodoVazio (pNodoA *arv*)

Testa se o nodo é vazio.

Parâmetros:

arv nó a ser testado

4.1.3.11 void printNiveis (pNodoA *arv*, int *count*)

Imprime a arvore com saída formatada em camadas.

Parâmetros:

arv raiz da arvore.

count valor da camada inicial.

4.1.3.12 pNodoA rotDireita (pNodoA arv)

Realizada rotacao a direita na arvore.

Parâmetros:

arv raiz da arvore.

4.1.3.13 pNodoA rotDupDireita (pNodoA arv)

Realizada rotacao dupla a direita na arvore.

Parâmetros:

arv raiz da arvore.

4.1.3.14 pNodoA rotDupEsquerda (pNodoA arv)

Realizada rotacao dupla a esquerda na arvore.

Parâmetros:

arv raiz da arvore.

4.1.3.15 pNodoA rotEsquerda (pNodoA arv)

Realizada rotacao a esquerda na arvore.

Parâmetros:

arv raiz da arvore.

4.2 Referência do Arquivo entry.h

Estruturas de Dados

- struct [entry_t](#)

4.2.1 Descrição Detalhada

Versão:

1.0

4.3 Referência do Arquivo lista.h

```
#include <stdio.h>
```

Estruturas de Dados

- struct `tac`
- struct `node_tac`

Funções

- struct `tac` * `create_inst_tac` (const char *res, const char *arg1, const char *op, const char *arg2)
Construtor de Instrucao TAC.
- void `print_inst_tac` (FILE *out, struct `tac` i)
Funcao que imprime o conteudo de uma instrucao TAC.
- void `print_tac` (FILE *out, struct `node_tac` *code)
Imprime no arquivo apontado por 'out' o conteudo da lista apontada por 'code'.
- void `append_inst_tac` (struct `node_tac` **code, struct `tac` *inst)
- void `cat_tac` (struct `node_tac` **code_a, struct `node_tac` **code_b)

4.3.1 Descrição Detalhada

4.3.2 Funções

4.3.2.1 void `append_inst_tac` (struct `node_tac` ** *code*, struct `tac` * *inst*)

Insere no fim da lista '*code*' o elemento '*inst*'.

Parâmetros:

code lista (possivelmente vazia) inicial, em entrada. Na saída, contém a mesma lista, com mais um elemento inserido no fim.

inst o elemento inserido no fim da lista.

4.3.2.2 void `cat_tac` (struct `node_tac` ** *code_a*, struct `node_tac` ** *code_b*)

Concatena a lista '*code_a*' com a lista '*code_b*'.

Parâmetros:

code_a lista (possivelmente vazia) inicial, em entrada. Na saída, contém a mesma lista concatenada com '*code_b*'.

code_b a lista concatenada com '*code_a*'.

4.3.2.3 `struct tac* create_inst_tac (const char * res, const char * arg1, const char * op, const char * arg2) [read]`

Construtor de Instrucao TAC. Para testes, pode-se usar qualquer string em argumentos.

Parâmetros:

res um char*.
arg1 um char*.
op um char*.
arg2 um char*.

Retorna:

um ponteiro sobre uma 'struct tac'.

4.3.2.4 `void print_inst_tac (FILE * out, struct tac i)`

Funcao que imprime o conteudo de uma instrucao TAC.

Parâmetros:

out um ponteiro sobre um arquivo (aberto) aonde ira ser escrita a instrucao.
i a instrucao a ser impressa.

4.3.2.5 `void print_tac (FILE * out, struct node_tac * code)`

Imprime no arquivo apontado por 'out' o conteudo da lista apontada por 'code'.

Parâmetros:

out um ponteiro sobre um arquivo (aberto) aonde ira ser escrita a lista (uma linha por elemento).
code o ponteiro para a lista a ser impressa.

4.4 Referência do Arquivo node.h

```
#include <stdio.h>
```

Estruturas de Dados

- struct [_node](#)

Definições e Macros

- #define [MAX_CHILDREN_NUMBER](#) 10

Constantes

Serie de constantes que serviraõ para definir tipos de nos (na arvore), a partir da etapa 4 - irrelevante por enquanto.

- #define [program_node](#) 299
- #define [idf_node](#) 300
- #define [int_node](#) 301
- #define [float_node](#) 302
- #define [str_node](#) 303
- #define [empty_node](#) 304
- #define [proc_node](#) 305
- #define [param_node](#) 306
- #define [decl_node](#) 307
- #define [decl_list_node](#) 308
- #define [op_node](#) 309
- #define [nop_node](#) 310
- #define [return_node](#) 311
- #define [if_node](#) 312
- #define [while_node](#) 313
- #define [print_node](#) 314
- #define [cond_node](#) 315
- #define [affect_node](#) 316
- #define [or_node](#) 317
- #define [and_node](#) 318
- #define [eq_node](#) 319
- #define [neq_node](#) 320
- #define [inf_node](#) 321
- #define [sup_node](#) 322
- #define [inf_eq_node](#) 323
- #define [sup_eq_node](#) 324
- #define [plus_node](#) 325
- #define [minus_node](#) 326
- #define [mult_node](#) 327
- #define [div_node](#) 328
- #define [mod_node](#) 329
- #define [umenos_node](#) 330
- #define [not_node](#) 331
- #define [char_node](#) 332
- #define [bloc_node](#) 333
- #define [true_node](#) 335
- #define [false_node](#) 336

Definições de Tipos

- typedef int `Node_type`
- typedef struct `_node` `Node`

Funções

- `Node * create_node` (int nl, `Node_type` t, char *lexema, void *att, int nbc, `Node **children`)
- `Node * create_leaf` (int nl, `Node_type` t, char *lexema, void *att)
- int `nb_of_children` (`Node *n`)
- `Node * child` (`Node *n`, int i)
- int `pack_nodes` (`Node ***array_of_nodes`, int cur_size, `Node *n`)
- int `deep_free_node` (`Node *n`)
- int `height` (`Node *n`)

4.4.1 Descrição Detalhada

Versão:

1.1

4.4.2 Definições e macros

4.4.2.1 #define MAX_CHILDREN_NUMBER 10

número máximo de filhos de uma struct `_node`

4.4.2.2 #define program_node 299

4.4.3 Definições dos tipos

4.4.3.1 typedef struct _node Node

Estrutura de dados parcial para o nó da árvore. Trata-se de uma árvore generalizada: qualquer nó pode ter de 0 até MAX_CHILDREN_NUMBER filhos.

4.4.3.2 typedef int Node_type

tipo do nó

4.4.4 Funções

4.4.4.1 Node* child (Node * n, int i)

accessor to the i'th child of a Node.

Parâmetros:

n the node to be consulted. Must return an error if 'n' is NULL.

i the number of the child that one wants. Must be strictly lower than `n->num_children` and larger than 0. Must return an error if *i* is not correct.

Retorna:

a pointer on a Node.

4.4.4.2 Node* create_leaf (int *nl*, Node_type *t*, char * *lexema*, void * *att*)

Constructor of a leaf Node (without any child).

Parâmetros:

nl line number of the instruction that originates the node.

t node type (one of the values # define'd above). Must return an error if the type is not correct.

lexema whatever string you want to associate to the node.

att a semantica attribute (can be NULL for now).

Retorna:

a (pointer) on a new Node.

4.4.4.3 Node* create_node (int *nl*, Node_type *t*, char * *lexema*, void * *att*, int *nbc*, Node ** *children*)

Constructor of a Node.

Parâmetros:

nl line number of the instruction that originates the node.

t node type (one of the values # define'd above). Must return an error if the type is not correct.

lexema whatever string you want to associate to the node.

att a semantical attribute.

nbc number of children nodes ($\leq \text{MAX_CHILDREN_NUMBER}$ and ≥ 0). Must return an error if *nbc* is not correct.

children array of children nodes (of size '*nbc*').

Retorna:

a (pointer on a) new Node.

4.4.4.4 int deep_free_node (Node * *n*)

Destructor of a Node. Deallocates (recursively) all the tree rooted at '*n*'.

Parâmetros:

n the root node

4.4.4.5 int height (Node * *n*)

returns the height of the tree rooted by '*n*'. The height of a leaf is 1.

Parâmetros:

n the root node

Retorna:

the height of the tree

4.4.4.6 int nb_of_children (Node * *n*)

accessor to the number of children of a Node.

Parâmetros:

n node

Retorna:

the number of children

Aviso:

n shouldn't be NULL

4.4.4.7 int pack_nodes (Node *** *array_of_nodes*, int *cur_size*, Node * *n*)

Pushes '*n*' Nodes on bottom of an array of Node* which contains originally '*cur_size*' entries. Returns the new number of entries (ie '*cur_size*+1') in the array '*array_of_nodes*'. This function is convenient to be used with create_node (see its last argument). On the first call, '*array_of_nodes*' should not be allocated, and '*cur_size*' should be zero. Implementation limit: since these Node* structures are meant to be used in a compiler, one expects an upper limit of MAX_CHILDREN_NUMBER nodes to be packed. Trying to pack more than this limit must raise an error. Typical use: Node** children; pf1 = create_leaf(1, int_node, "1", NULL); pf3 = create_leaf(1, int_node, "2", NULL); pack_nodes(&children, 0, pf1); pack_nodes(&children, 1, pf3);

Parâmetros:

array_of_nodes the array where the node will be pushed

cur_size the current size of the array

n the node to be pushed

Retorna:

the new number of entries (*cur_size* + 1)

Aviso:

cur_size should be less than MAX_CHILDREN_NUMBER

4.5 Referência do Arquivo symbol_table.h

```
#include <stdio.h>
#include "entry.h"
#include "avl.h"
```

Estruturas de Dados

- struct [s_symbol_t](#)
Encapsulacao de um tipo abstrato que se chamara 'symbol_t'.

Definições e Macros

- #define [TABLE_SIZE](#) 1000000

Definições de Tipos

- typedef struct [s_symbol_t](#) * [symbol_t](#)

Funções

- int [init_table](#) ([symbol_t](#) *table)
Inicializar a tabela de Hash.
- void [free_table](#) ([symbol_t](#) *table)
Destruir a tabela de Hash.
- [entry_t](#) * [lookup](#) ([symbol_t](#) table, char *name)
Retornar um ponteiro sobre a entrada associada a 'name'.
- int [insert](#) ([symbol_t](#) *table, [entry_t](#) *entry)
Inserir uma entrada em uma tabela.
- int [print_table](#) ([symbol_t](#) table)
Imprimir o conteudo de uma tabela.
- int [print_file_table](#) (FILE *out, [symbol_t](#) table)
Imprimir o conteudo de uma tabela em um arquivo.
- unsigned long [hash](#) (char *str)
funcao de hash

4.5.1 Descrição Detalhada

Versão:

1.1

4.5.2 Definições e macros

4.5.2.1 #define TABLE_SIZE 1000000

tamanho da tabela hash

4.5.3 Definições dos tipos

4.5.3.1 typedef struct s_symbol_t * symbol_t

a pointer to a symbol_table

4.5.4 Funções

4.5.4.1 void free_table (symbol_t * table)

Destruir a tabela de Hash. 'free_table' eh o destrutor da estrutura de dados. Deve ser chamado pelo usuario no fim de seu uso de uma tabela de simbolos.

Parâmetros:

table uma referencia sobre uma tabela de simbolos.

4.5.4.2 unsigned long hash (char * str)

funcao de hash djb2 - do site <http://www.cse.yorku.ca/~oz/hash.html>

Parâmetros:

str string para se calcular o valor hash

Retorna:

valor hash da string

4.5.4.3 int init_table (symbol_t * table)

Inicializar a tabela de Hash.

Parâmetros:

table uma referencia sobre uma tabela de simbolos.

Retorna:

o valor 0 se deu certo.

4.5.4.4 `int insert (symbol_t * table, entry_t * entry)`

Inserir uma entrada em uma tabela.

Parâmetros:

table uma tabela de símbolos.

entry uma entrada.

Retorna:

um número negativo se não se conseguiu efetuar a inserção, zero se deu certo.

4.5.4.5 `entry_t* lookup (symbol_t table, char * name)`

Retornar um ponteiro sobre a entrada associada a 'name'. Essa função deve consultar a tabela de símbolos para verificar se se encontra nela uma entrada associada a um `char*` (string) fornecido em entrada. Para a implementação, será necessário usar uma função que mapeia um `char*` a um número inteiro. Aconselha-se, por exemplo, consultar o livro do dragão (Aho/Sethi/Ulman), Fig. 7.35 e a função `HPJW`.

Parâmetros:

table uma tabela de símbolos.

name um `char*` (string).

Retorna:

um ponteiro sobre a entrada associada a 'name', ou `NULL` se 'name' não se encontrou na tabela.

4.5.4.6 `int print_file_table (FILE * out, symbol_t table)`

Imprimir o conteúdo de uma tabela em um arquivo. A formatação exata é deixada a cargo do programador. Deve-se listar todas as entradas contidas na tabela através de seu nome (`char*`). Deve retornar o número de entradas na tabela. A saída deve ser dirigida para um arquivo, cujo descritor é passado em parâmetro.

Parâmetros:

out um descritor de arquivo (`FILE*`).

table uma tabela de símbolos.

Retorna:

o número de entradas na tabela.

4.5.4.7 `int print_table (symbol_t table)`

Imprimir o conteúdo de uma tabela. A formatação exata é deixada a cargo do programador. Deve-se listar todas as entradas contidas na tabela através de seu nome (`char*`). Deve retornar o número de entradas na tabela.

Parâmetros:

table uma tabela de símbolos.

Retorna:

o numero de entradas na tabela.

Índice Remissivo

- [_node](#), [5](#)
 - [attribute](#), [5](#)
 - [child](#), [5](#)
 - [id](#), [5](#)
 - [lexeme](#), [5](#)
 - [num_children](#), [6](#)
 - [num_line](#), [6](#)
 - [type](#), [6](#)
- [altura](#)
 - [avl.h](#), [14](#)
- [append_inst_tac](#)
 - [lista.h](#), [19](#)
- [arg1](#)
 - [tac](#), [11](#)
- [arg2](#)
 - [tac](#), [11](#)
- [attribute](#)
 - [_node](#), [5](#)
- [avl.h](#), [13](#)
 - [altura](#), [14](#)
 - [balanceamentoDir](#), [15](#)
 - [balanceamentoEsq](#), [15](#)
 - [calculaFatorB](#), [15](#)
 - [destroiAVL](#), [15](#)
 - [findNodo](#), [15](#)
 - [imprimeAVL](#), [15](#)
 - [inicializaAVL](#), [16](#)
 - [insereAVL](#), [16](#)
 - [NodoVazio](#), [16](#)
 - [pNodoA](#), [14](#)
 - [printNiveis](#), [16](#)
 - [rotDireita](#), [16](#)
 - [rotDupDireita](#), [17](#)
 - [rotDupEsquerda](#), [17](#)
 - [rotEsquerda](#), [17](#)
- [balanceamentoDir](#)
 - [avl.h](#), [15](#)
- [balanceamentoEsq](#)
 - [avl.h](#), [15](#)
- [calculaFatorB](#)
 - [avl.h](#), [15](#)
- [cat_tac](#)
 - [lista.h](#), [19](#)
- [child](#)
 - [_node](#), [5](#)
 - [node.h](#), [22](#)
- [create_inst_tac](#)
 - [lista.h](#), [19](#)
- [create_leaf](#)
 - [node.h](#), [23](#)
- [create_node](#)
 - [node.h](#), [23](#)
- [deep_free_node](#)
 - [node.h](#), [23](#)
- [desloc](#)
 - [entry_t](#), [7](#)
- [destroiAVL](#)
 - [avl.h](#), [15](#)
- [dir](#)
 - [hash_entry_t](#), [8](#)
- [entry](#)
 - [hash_entry_t](#), [8](#)
 - [s_symbol_t](#), [10](#)
- [entry.h](#), [18](#)
- [entry_t](#), [7](#)
 - [desloc](#), [7](#)
 - [extra](#), [7](#)
 - [name](#), [7](#)
 - [size](#), [7](#)
 - [type](#), [7](#)
- [esq](#)
 - [hash_entry_t](#), [8](#)
- [extra](#)
 - [entry_t](#), [7](#)
- [fatorB](#)
 - [hash_entry_t](#), [8](#)
- [findNodo](#)
 - [avl.h](#), [15](#)
- [free_table](#)
 - [symbol_table.h](#), [26](#)
- [hash](#)
 - [symbol_table.h](#), [26](#)
- [hash_entry_t](#), [8](#)

- dir, 8
- entry, 8
- esq, 8
- fatorB, 8
- height
 - node.h, 23
- id
 - _node, 5
- imprimeAVL
 - avl.h, 15
- inicializaAVL
 - avl.h, 16
- init_table
 - symbol_table.h, 26
- insereAVL
 - avl.h, 16
- insert
 - symbol_table.h, 26
- inst
 - node_tac, 9
- lexeme
 - _node, 5
- lista.h, 19
 - append_inst_tac, 19
 - cat_tac, 19
 - create_inst_tac, 19
 - print_inst_tac, 20
 - print_tac, 20
- lookup
 - symbol_table.h, 27
- MAX_CHILDREN_NUMBER
 - node.h, 22
- name
 - entry_t, 7
- nb_of_children
 - node.h, 24
- next
 - node_tac, 9
- Node
 - node.h, 22
- node.h, 21
 - child, 22
 - create_leaf, 23
 - create_node, 23
 - deep_free_node, 23
 - height, 23
 - MAX_CHILDREN_NUMBER, 22
 - nb_of_children, 24
 - Node, 22
 - Node_type, 22
 - pack_nodes, 24
 - program_node, 22
- node_tac, 9
 - inst, 9
 - next, 9
 - prev, 9
- Node_type
 - node.h, 22
- NodoVazio
 - avl.h, 16
- num_children
 - _node, 6
- num_line
 - _node, 6
- op
 - tac, 11
- pack_nodes
 - node.h, 24
- pNodoA
 - avl.h, 14
- prev
 - node_tac, 9
- print_file_table
 - symbol_table.h, 27
- print_inst_tac
 - lista.h, 20
- print_table
 - symbol_table.h, 27
- print_tac
 - lista.h, 20
- printNiveis
 - avl.h, 16
- program_node
 - node.h, 22
- res
 - tac, 11
- rotDireita
 - avl.h, 16
- rotDupDireita
 - avl.h, 17
- rotDupEsquerda
 - avl.h, 17
- rotEsquerda
 - avl.h, 17
- s_symbol_t, 10
 - entry, 10
- size
 - entry_t, 7
- symbol_t
 - symbol_table.h, 26

- symbol_table.h, [25](#)
 - free_table, [26](#)
 - hash, [26](#)
 - init_table, [26](#)
 - insert, [26](#)
 - lookup, [27](#)
 - print_file_table, [27](#)
 - print_table, [27](#)
 - symbol_t, [26](#)
 - TABLE_SIZE, [26](#)
- TABLE_SIZE
 - symbol_table.h, [26](#)
- tac, [11](#)
 - arg1, [11](#)
 - arg2, [11](#)
 - op, [11](#)
 - res, [11](#)
- type
 - _node, [6](#)
 - entry_t, [7](#)