Pico

1.0

Gerado por Doxygen 1.6.1

Mon Apr 12 21:14:31 2010

Sumário

Capítulo 1

Índice das Estruturas de Dados

1.1 Estruturas de Dados

Aqui estão as estruturas de dados e suas respectivas descrições:

Capítulo 2

Índice dos Arquivos

2.1 Lista de Arquivos

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

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 no da arvore. Trata-se de uma arvore generalizada: qualquer no pode ter de 0 ateh 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 nodos.

3.1.2.3 int _node::id

rótulo do nó. Cada no 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)

  Inserve 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 arvore.

Parâmetros:

arv raiz da arvore.

4.1.3.3 pNodoA balanceamentoEsq (pNodoA *arv*)

Realizada balanceamento a esquerda na arvore.

Parâmetros:

arv raiz da arvore.

4.1.3.4 int calculaFatorB (pNodoA *arv*)

Calcula o fator de balanceamento da arvore.

Parâmetros:

arv raiz da árvore

4.1.3.5 void destroiAVL (pNodoA *arv*)

Destroi a arvore, desalocando memoria ocupada por ela.

Parâmetros:

arv raiz da arvore a ser destruida.

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

Encontra um elemento na arvore.

Parâmetros:

arv raiz da arvore.

name nome do elemento a ser pesquisado.

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

Retorna:

ponteiro para o nodo da arvore que contem 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 saida 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 de o nodo é vazio.

Parâmetros:

arv nó a ser testado

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

Imprime a arvore com saida 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 saida, contem 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 saida, contem 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 servirao 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 no da arvore. Trata-se de uma arvore generalizada: qualquer no pode ter de 0 ateh 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 en 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 in 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 in not correct.

lexema whatever string you want to associate to the node.

att a semantical attribute.

nbc number of children nodes (<= MAX_CHILDREN_NUMBER and >= 0). Must return an error if nbc it 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. Desallocates (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 (cursize + 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 simbolos.

entry uma entrada.

Retorna:

um numero negativo se nao se conseguiu efetuar a insercao, 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 funcao deve consultar a tabela de simbolos para verificar se se encontra nela uma entrada associada a um char∗ (string) fornecido em entrada. Para a implementacao, sera necessario usar uma funcao que mapeia um char∗ a um numero inteiro. Aconselha-se, por exemplo, consultar o livro do dragao (Aho/Sethi/Ulman), Fig. 7.35 e a funcao HPJW.

Parâmetros:

table uma tabela de simbolos.

name um char∗ (string).

Retorna:

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

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

Imprimir o conteudo de uma tabela em um arquivo. A formatacao exata e deixada a carga do programador. Deve-se listar todas as entradas contidas na tabela atraves de seu nome (char∗). Deve retornar o numero de entradas na tabela. A saida deve ser dirigida para um arquivo, cujo descritor e passado em parametro.

Parâmetros:

out um descrito de arquivo (FILE∗).

table uma tabela de simbolos.

Retorna:

o numero de entradas na tabela.

4.5.4.7 int print_table (symbol_t *table*)

Imprimir o conteudo de uma tabela. A formatacao exata e deixada a carga do programador. Deve-se listar todas as entradas contidas na tabela atraves de seu nome (char∗). Deve retornar o numero de entradas na tabela.

Parâmetros:

table uma tabela de simbolos.

Retorna:

o numero de entradas na tabela.

Índice Remissivo