Estrutura de Dados - Trabalho 1 Guia de Referência

1.0

Gerado por Doxygen 1.3.7

Tue Jun 7 22:36:05 2005

Contents

1 Estrutura de Dados - Trabalho 1

Ciência da Computação - UFES 2005/01

1.1 Descrição do trabalho

Esse trabalho tem como objetivo apresentar uma implementação das seguintes estruturas lineares:

- Fila
- Lista
- Lista Ordenada
- Pilha
- Tabela Hash

Para tal, foi desenvolvido um aplicativo para explorar essas estruturas de maneira genérica.

A linguagem adotada para a implementação do trabalho foi o **C++**, utilizando o paradigma de programação orientada a objetos e o recurso de *templates*. Esta documentação foi gerada com o auxílio da ferramenta Doxygen < `http://www.doxygen.org` >

Todo o trabalho é disponibilizado sob os termos da licensa **GPL**. (veja docs/LICENSE)

1.1.1 Autor

Reinaldo de Souza Junior < `juniorz@phpbb.com.br` >

1.1.2 Professor

Fabrício V. Matos < `fabricio@qualidata.com.br` >

2 Estrutura de Dados - Trabalho 1 Índice Hierárquico

2.1 Estrutura de Dados - Trabalho 1 Hierarquia de Classes

Esta lista de hierarquias está parcialmente ordenada (ordem alfabética):

3 Estrutura de Dados - Trabalho 1 Índice das Estruturas de Dados

3.1 Estrutura de Dados - Trabalho 1 Estruturas de Dados

Aqui estão as estruturas de dados e suas respectivas descrições:

4 Estrutura de Dados - Trabalho 1 Estruturas

4.1 Referência da Classe Pessoa

`#include <Pessoa.h>`

4.1.1 Descrição Detalhada

Pessoa genérica para exemplificar a aplicação das estruturas de dados.

Definição na linha 17 do arquivo Pessoa.h.

Métodos Públicos

- char * GetNome ()
  Obtém o nome da pessoa.
- int GetCodigo ()
  Obtém o Código da pessoa.
- void SetNome (char *nome)
  Define o nome da pessoa.
- void SetCodigo (int codigo)
  Define o Código da pessoa.
- operator const int () const
  Faz a conversão de Pessoa para (int).
- operator const float () const
  Faz a conversão de Pessoa para (float).

Atributos Privados

- int codPessoa
  Código da Pessoa.
- char strNome [255]
  Nome da Pessoa.

Amigas

- bool operator> (const Pessoa &a, const Pessoa &b)
  Sobreescreve o operador >.
- bool operator< (const Pessoa &a, const Pessoa &b)
  Sobreescreve o operador <.
- int operator== (const Pessoa &a, const Pessoa &b)
  Sobreescreve o operador ==.
- bool operator>= (const Pessoa &a, const Pessoa &b)
  Sobreescreve o operador >=.
- bool operator<= (const Pessoa &a, const Pessoa &b)
  Sobreescreve o operador <=.
- ostream & operator<< (ostream &out, const Pessoa &a)
  Sobreescreve o operador << (para um stream de saida).

4.1.2 Métodos

4.1.2.1 char* Pessoa::GetNome () [inline]

Obtém o nome da pessoa.

Retorna:
Nome da Pessoa

Definição na linha 31 do arquivo Pessoa.h.

```
31 { return strNome; };
```

4.1.2.2 int Pessoa::GetCodigo () [inline]

Obtém o Código da pessoa.

Retorna:
Código da Pessoa

Definição na linha 37 do arquivo Pessoa.h.

```
37 { return codPessoa; };
```

4.1.2.3 void Pessoa::SetNome (char * *nome*) [inline]

Define o nome da pessoa.

Parâmetros:
nome Nome da Pessoa

Definição na linha 43 do arquivo Pessoa.h.

Referenciado por SetNome().

```
43 { strncpy(strNome, nome, 255); };
```

4.1.2.4 void Pessoa::SetCodigo (int *codigo*) [inline]

Define o Código da pessoa.

Parâmetros:
codigo Código da Pessoa

Definição na linha 49 do arquivo Pessoa.h.

Referenciado por SetCodigo().

```
49 { codPessoa = codigo; };
```

4.1.2.5 Pessoa::operator const int () const [inline]

Faz a conversão de Pessoa para (int).

Definição na linha 84 do arquivo Pessoa.h.

```
84 { return codPessoa; };
```

4.1.2.6 Pessoa::operator const float () const [inline]

Faz a conversão de Pessoa para (float).

Definição na linha 89 do arquivo Pessoa.h.

```
89 { return (float) codPessoa; };
```

4.1.3 Amigas e Funções Relacionadas

4.1.3.1 bool operator> (const Pessoa & *a*, const Pessoa & *b*) [friend]

Sobreescreve o operador >.

Definição na linha 54 do arquivo Pessoa.h.

```
54 { return a.codPessoa > b.codPessoa; };
```

4.1.3.2 bool operator< (const Pessoa & *a*, const Pessoa & *b*) [friend]

Sobreescreve o operador <.

Definição na linha 59 do arquivo Pessoa.h.

```
59 { return a.codPessoa < b.codPessoa; };
```

4.1.3.3 int operator== (const Pessoa & *a*, const Pessoa & *b*) [friend]

Sobreescreve o operador ==.

Definição na linha 64 do arquivo Pessoa.h.

```
64 { return a.codPessoa == b.codPessoa; };
```

4.1.3.4 bool operator>= (const Pessoa & *a*, const Pessoa & *b*) [friend]

Sobreescreve o operador >=.

Definição na linha 69 do arquivo Pessoa.h.

```
69 { return a.codPessoa >= b.codPessoa; };
```

4.1.3.5 bool operator<= (const Pessoa & *a*, const Pessoa & *b*) [friend]

Sobreescreve o operador <=.

Definição na linha 74 do arquivo Pessoa.h.

```
74 { return a.codPessoa <= b.codPessoa; };
```

4.1.3.6 ostream& operator<< (ostream & *out*, const Pessoa & *a*) [friend]

Sobreescreve o operador << (para um stream de saida).

Definição na linha 79 do arquivo Pessoa.h.

```
79 { out << a.codPessoa; out << " - "; out << a.strNome;  };
```

4.1.4 Campos e Atributos

4.1.4.1 int Pessoa::codPessoa [private]

Código da Pessoa.

Definição na linha 19 do arquivo Pessoa.h.

Referenciado por GetCodigo(), operator const float(), operator const int() e SetCodigo().

4.1.4.2 char Pessoa::strNome[255] [private]

Nome da Pessoa.

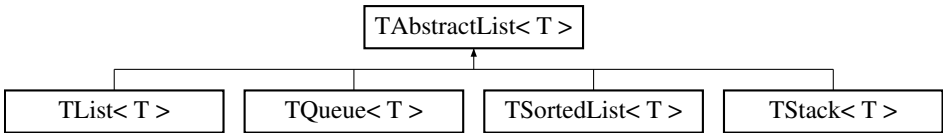
Definição na linha 20 do arquivo Pessoa.h.

Referenciado por GetNome() e SetNome().

4.2 Referência da Template de Classe TAbstractList< T >

`#include <TAbstractList.h>`

Diagrama de Hierarquia para TAbstractList< T >::



4.2.1 Descrição Detalhada

template<class T> class TAbstractList< T >

Lista Abstrata.

Classe abstrata da qual a maioria das outras estruturas lineares derivam.

Versão:
1.0.1

Autor:
Reinaldo de Souza Junior <juniorz@gmail.com.br>

Data:
05/2005

Definição na linha 62 do arquivo TAbstractList.h.

Métodos Protegidos

- TAbstractList ()
  Construtor da Lista Abstrata.
- ~TAbstractList ()
  Destrutor da Lista Abstrata.
- int Count ()

Retorna o numero de elementos da estrutura.

- T * Get (int pos)
  Retorna um elemento de uma posição específica.
- T * GetCurrent ()
  Retorna o elemento corrente da estrutura.
- int Insert (T *data, int pos)
  Insere um elemento em uma posição específica.
- int Del (int pos)
  Remove um elemento da estrutura.
- void DelAll ()
  Remove todos os elemento da estrutura.
- int Kill (int pos)
  Remove e destrói um elemento da estrutura.
- void KillAll ()
  Remove e destrói todos os elemento da estrutura.
- int MoveFirst ()
  Aponta o elemento corrente para o primeiro elemento da estrutura.
- int MoveNext ()
  Aponta o elemento corrente para próximo elemento da estrutura.
- int MovePrior ()
  Aponta o elemento corrente para o elemento anterior da estrutura.
- int MoveLast ()
  Aponta o elemento corrente para o último elemento da estrutura.
- int GoTo (int pos)
  Aponta o elemento corrente para o elemento na posição `pos`.
- void ResetCurrent ()
  Reseta o elemento corrente da estrutura.
- int Search (T *procurado)
  Busca um elemento na estrutura.
- void Sort ()
  Ordena a estrutura em ordem crescente.

Atributos Protegidos

- int Size

  Número de elementos que a lista possui.

- int PosCorrente

  Posição do elemento apontado por TAbstractList::pFirst.

- Node * pFirst

  Apontador para o primeiro elemento da lista.

- Node * pCurrent

  Apontador para o elemento corrente da lista.

- Node * pLast

  Apontador para o último elemento da lista.

Tipos Privados

- typedef TAbstractList::node Node

  Um Nó que representa um elemento da lista.

Métodos Privados

- Node * _CreateNode (T *dado, Node *anterior, Node *posterior)

  Cria um novo elemento para a estrutura.

- bool _NodeSetPrior (Node *node, Node *anterior)

  Define o pPrior de um nó (caso ele realmente seja um nó).

- bool _NodeSetNext (Node *node, Node *proximo)

  Define o pNext de um nó (caso ele realmente seja um nó).

- bool _ToNext ()

  Aponta para o próximo elemento e atualiza a posição.

- bool _ToPrior ()

  Aponta para o próximo elemento e atualiza a posição.

- Node * _PrepareDel (int pos)

  Prepara a lista para um Del.

- void _BubbleSort ()

  Executa um Bubblesort na lista.

- void _Swap (Node *a, Node *b)

  Troca a posicao de 2 elementos.

4.2.2 Definições de Tipos

4.2.2.1 template<class T> typedef struct TAbstractList::node TAbstractList< T >::Node [private]

Um Nó que representa um elemento da lista.

4.2.3 Construtores & Destrutores

4.2.3.1 template<class T> TAbstractList< T >::TAbstractList () [inline, protected]

Construtor da Lista Abstrata.

Instancia um novo objeto TAbstractList

Complexidade: $O(1)$

Definição na linha 107 do arquivo TAbstractList.h.

```
107 : PosCorrente(0), pFirst(NULL), pCurrent(NULL), pLast(NULL), Size(0) {  };
```

4.2.3.2 template<class T> TAbstractList< T >::~TAbstractList () [inline, protected]

Destrutor da Lista Abstrata.

Esse método também limpa a estrutura.

Complexidade: $O(n)$

Definição na linha 120 do arquivo TAbstractList.h.

```
120 { DelAll(); };
```

4.2.4 Métodos

4.2.4.1 template<class T> int TAbstractList< T >::Count () [inline, protected]

Retorna o numero de elementos da estrutura.

Complexidade: $O(1)$

Reimplementado por TList< T >, TQueue< T >, TSortedList< T > e TStack< T >.

Definição na linha 132 do arquivo TAbstractList.h.

Referenciado por TStack< T >::Count(), TSortedList< T >::Count(), TQueue< T >::Count(), TList< T >::Count(), TSortedList< T >::Insert() e TStack< T >::Pop().

```
132 { return Size; };
```

4.2.4.2 template<class T> T* TAbstractList< T >::Get (int *pos*) [inline, protected]

Retorna um elemento de uma posição específica.

Retorna um ponteiro para elemento da estrutura que ocupa a posição `pos`

Parâmetros:

pos Posição do elemento que se deseja obter

Retorna:

Ponteiro associado ao nó que ocupa a posição `pos` na estrutura

Complexidade: $O(n)$

Reimplementado por TList< T >, TQueue< T >, TSortedList< T > e TStack< T >.

Definição na linha 149 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::Get() e TAbstractList< T >::Insert().

```
149 { try { GoTo(pos); } catch (...) { throw; } return GetCurrent(); };
```

4.2.4.3 template<class T> T* TAbstractList< T >::GetCurrent () [inline, protected]

Retorna o elemento corrente da estrutura.

Retorna:

Ponteiro associado ao nó que ocupa a posição corrente na estrutura.

Complexidade: $O(1)$

Reimplementado por TList< T >, TQueue< T > e TSortedList< T >.

Definição na linha 162 do arquivo TAbstractList.h.

Referenciado por TStack< T >::Base(), TAbstractList< T >::Get(), TSortedList< T >::GetCurrent(), TQueue< T >::GetCurrent(), TList< T >::GetCurrent() e TStack< T >::Top().

```
162 { if(pCurrent == NULL) { throw new TENullElement("TAbstractList::GetCurrent"); } return pCurrer
```

4.2.4.4 template<class T> int TAbstractList< T >::Insert (T * *data*, int *pos*) [protected]

Insere um elemento em uma posição específica.

Insere um elemento que aponta para `data` na posição `pos` da estrutura.

Parâmetros:

data Ponteiro para um elemento que será inserido na estrutura

pos Posição onde o elemento deve ser inserido

Retorna:

Posição onde o elemento foi inserido

Exceções:

TENullElement*∗* caso `data` seja nulo (não aponta para nada). **Veja:** TENullElement

TEOutOfBounds*∗* caso `pos` seja uma posição inválida. **Veja:** TEOutOfBounds

Pós-Condição:

```
data == Get(pos);
```

Complexidade: $O(n)$

Reimplementado por TList< T >.

Definição na linha 751 do arquivo TAbstractList.h.

```
752 {
753         Node *prior, *pNew;
754 
755         //Inserindo u melemento nulo
756         if(data == NULL)
757                 throw new TENullElement("TAbstractList::Insert");
758 
759         //Movimenta
760         GoTo(i);
761 
762         //Define o nó anterior
763         prior = (pCurrent == NULL) ? pCurrent : pCurrent->pPrior;
764 
765         //Cria o novo nó
766         pNew = _CreateNode(data, prior, pCurrent);
767 
768         /*************************************
769         * Essa ordem de avaliação é importante
770         *************************************/
771 
772         //Ultimo elemento
773         if(i == Size) {
774                 pNew->pPrior = pCurrent;
775                 pNew->pNext = NULL;
776 
777                 pLast = pNew;   //Atualiza o ultimo
778         }
779         else {
```

```
780                 pNew->pPrior = pCurrent->pPrior;
781                 pCurrent->pPrior = pNew;
782                 //pNew->pNext = pCurrent (por definicao)
783         }
784
785         //Primeiro elemento
786         if(i == 0) {
787                 pNew->pPrior = NULL;
788                 //pNew->pNext = pCurrent (por definicao)
789
790                 pFirst = pNew;  //Atualiza o primeiro
791         }
792         else {
793                 pNew->pPrior->pNext = pNew;
794         }
795
796         //O elemento sendo inserido nao é o unico
797         if(pCurrent != NULL) {
798                 //Primeiro
799                 if(i == 0)
800                 pCurrent->pPrior = pNew;
801
802                 //Ultimo
803                 if(i == Size)
804                 pCurrent->pNext = pNew;
805         }
806
807         //Atualiza o elemento corrente
808         pCurrent = pNew;
809         PosCorrente = i;
810
811         //Atualiza o tamanho
812         Size++;
813
814 #ifdef DEBUG
815         //Invariante
816         INVARIANTE(data == Get(i), "data == Get(i) | from TAbstractList::Insert ")
817 #endif
818
819         //Retorna a posição
820         return i;
821 }
```

4.2.4.5 template<class T> int TAbstractList< T >::Del (int *pos*) [protected]

Remove um elemento da estrutura.

Remove o elemento da estrutura na posição `pos`. Esse método não destrói o elemento apontado pelo nó, apenas retira-o da estrutura, para destruí-lo use Kill()

Parâmetros:

pos Posição do elemento a ser removido da lista.

Retorna:

Posição do elemento corrente da estrutura após a remoção.

Complexidade: $O(n)$

Reimplementado por TList< T > e TSortedList< T >.

Definição na linha 836 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::DelAll().

```
837 {
838         //Organiza a lista para a remoção e obtem o pCurrent
839         Node* pProximoCurrent = _PrepareDel(posicao);
840 
841         //Remove o elemento da lista
842         delete pCurrent;
843 
844         //Atualiza o pCurrent (evita um novo GoTo)
845         pCurrent = pProximoCurrent;
846 
847         //Decrementa o tamanho
848         Size--;
849 
850         return PosCorrente;
851 }
```

4.2.4.6 template<class T> void TAbstractList< T >::DelAll () [protected]

Remove todos os elemento da estrutura.

Complexidade: $O(n)$

Reimplementado por TList< T >, TQueue< T >, TSortedList< T > e TStack< T >.

Definição na linha 864 do arquivo TAbstractList.h.

Referenciado por TStack< T >::DelAll(), TSortedList< T >::DelAll(), TQueue< T >::DelAll(), TList< T >::DelAll() e TAbstractList< T >::~TAbstractList().

```
865 {
866         //Preferi utilizar um loop em conjunto com Del() por motivos de:
867         //1) Elegância
868         //2) A complexidade continua sendo O(n)
869 
870         // Vai para o primeiro
871         MoveFirst();    //O(1)
872 
873         // n vezes
874         while(pCurrent != NULL)
875                 Del(0);         // Remove o primeiro
876                                 // O(1)
877 }
```

4.2.4.7 template<class T> int TAbstractList< T >::Kill (int *pos*) [protected]

Remove e destrói um elemento da estrutura.

Libera a memória ocupada pelo elemento na posição `pos` e retira-o da estrutura. Após a chamada desse método o elemento, qualquer referência ao elemento destruído será inválida.

Parâmetros:

pos Posição do elemento a ser destruído.

Retorna:

Posição do elemento corrente da estrutura.

Complexidade: $O(n)$

Reimplementado por TList< T > e TSortedList< T >.

Definição na linha 884 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::KillAll().

```
885 {
886         //Organiza a lista para a remoção e obtem o pCurrent
887         Node* pProximoCurrent = _PrepareDel(posicao);
888 
889         //Remove o elemento da lista
890         free(pCurrent->Data);
891         delete pCurrent;
892 
893         //Atualiza o pCurrent (evita um novo GoTo)
894         pCurrent = pProximoCurrent;
895 
896         //Decrementa o tamanho
897         Size--;
898 
899         return PosCorrente;
900 }
```

4.2.4.8 template<class T> void TAbstractList< T >::KillAll () [protected]

Remove e destrói todos os elemento da estrutura.

Complexidade: $O(n)$

Reimplementado por TList< T >, TQueue< T >, TSortedList< T > e TStack< T >.

Definição na linha 907 do arquivo TAbstractList.h.

Referenciado por TStack< T >::KillAll(), TSortedList< T >::KillAll(), TQueue< T >::KillAll() e TList< T >::KillAll().

```
908 {
909         //Preferi utilizar um loop em conjunto com Del() por motivos de:
910         //1) Elegância
911         //2) A complexidade continua sendo O(n)
912 
913         // Vai para o primeiro
914         MoveFirst();            // O(1)
915 
916         // n vezes
917         while(pCurrent != NULL)
918                 Kill(0);        // Remove o primeiro
919                                 // O(1)
920 }
```

4.2.4.9 template<class T> int TAbstractList< T >::MoveFirst () [inline, protected]

Aponta o elemento corrente para o primeiro elemento da estrutura.

Retorna:
Posição do elemento corrente da estrutura (usualmente retorna 0)

Complexidade: $O(1)$

Reimplementado por TList< T >, TQueue< T > e TSortedList< T >.

Definição na linha 259 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::DelAll(), TAbstractList< T >::GoTo(), TAbstractList< T >::KillAll(), TSortedList< T >::MoveFirst(), TQueue< T >::MoveFirst(), TList< T >::MoveFirst(), TAbstractList< T >::MoveNext(), TAbstractList< T >::Search(), TAbstractList< T >::Sort() e TStack< T >::Top().

```
259 { PosCorrente = (Size > 0) ? 0 : -1; pCurrent = pFirst; return PosCorrente; };
```

4.2.4.10 template<typename T> int TAbstractList< T >::MoveNext () [protected]

Aponta o elemento corrente para próximo elemento da estrutura.

Complexidade: $O(1)$

Reimplementado por TList< T >, TQueue< T > e TSortedList< T >.

Definição na linha 654 do arquivo TAbstractList.h.

Referenciado por TSortedList< T >::MoveNext(), TQueue< T >::MoveNext(), TList< T >::MoveNext(), TAbstractList< T >::Search() e TAbstractList< T >::Sort().

```
655 {
656         //Já está resetado
657         if(pCurrent == NULL)
658                 //Estou certo disso (especificação)
659                 return MoveFirst();
660
661         //Ao tentar passar do ultimo, reseta
662         if(! _ToNext() )
663                 ResetCurrent();
664
665         return PosCorrente;
666 }
```

4.2.4.11 template<typename T> int TAbstractList< T >::MovePrior () [protected]

Aponta o elemento corrente para o elemento anterior da estrutura.

Complexidade: $O(1)$

Reimplementado por TList< T >, TQueue< T > e TSortedList< T >.

Definição na linha 673 do arquivo TAbstractList.h.

Referenciado por TSortedList< T >::MovePrior(), TQueue< T >::MovePrior() e TList< T >::MovePrior().

```
674 {
675         //Já está resetado
676         if(pCurrent == NULL)
677                 //Estou certo disso (especificação)
678                 return MoveLast();
679
680         //Ao tentar ir para anter do primeiro, reseta
681         if(! _ToPrior() )
682                 ResetCurrent();
683
684         return PosCorrente;
685 }
```

4.2.4.12 template<class T> int TAbstractList< T >::MoveLast () [inline, protected]

Aponta o elemento corrente para o último elemento da estrutura.

Complexidade: $O(1)$

Reimplementado por TList< T >, TQueue< T > e TSortedList< T >.

Definição na linha 295 do arquivo TAbstractList.h.

Referenciado por TStack< T >::Base(), TAbstractList< T >::GoTo(), TSortedList< T >::MoveLast(), TQueue< T >::MoveLast(), TList< T >::MoveLast() e TAbstractList< T >::MovePrior().

```
295 { PosCorrente = Size - 1; pCurrent = pLast; return Size; };
```

4.2.4.13 template<typename T> int TAbstractList< T >::GoTo (int *pos*) [protected]

Aponta o elemento corrente para o elemento na posição `pos`.

Parâmetros:

pos Posição do novo elemento corrente

Retorna:

Posição do novo elemento corrente

Pós-Condição:

PosCorrente = `pos`

Exceções:

TEOutOfBounds caso `pos` seja uma posição inválida

Complexidade: $O(n)$

Reimplementado por TList< T > e TSortedList< T >.

Definição na linha 692 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_PrepareDel(), TAbstractList< T >::Get() e TAbstractList< T >::Insert().

```
693 {
694         //Fora dos limites
695         if(! (i >= 0 && i <= Size ) )
696                 throw new TEOutOfBounds("TAbstractList::GoTo");
697
698         //Movendo para alguma posicao de uma lista vazia
699         if(Size == 0 || i == PosCorrente)
700                 return PosCorrente;
701
702         //
703         //Otimiza a movimentação, restringindo o domínio
704         //para fazer um balanceamento das comparações
705         //
706
707         //Primeira metade
708         if(i <= Size / 2) {
709                 for(MoveFirst(); i > 0; i--)
710                         _ToNext();
711         }
712
713         //Segunda metade
714         else {
715                 for(i = Size - i, MoveLast(); i > 1; i--)
716                         _ToPrior();
717         }
718
719         return PosCorrente;
720 }
```

4.2.4.14 template<class T> void TAbstractList< T >::ResetCurrent () [inline, protected]

Reseta o elemento corrente da estrutura.

Complexidade: $O(1)$

Reimplementado por TList< T >, TQueue< T > e TSortedList< T >.

Definição na linha 326 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_BubbleSort(), TAbstractList< T >::MoveNext(), TAbstractList< T >::MovePrior(), TSortedList< T >::ResetCurrent(), TQueue< T >::ResetCurrent(), TList< T >::ResetCurrent() e TAbstractList< T >::Sort().

```
326 { pCurrent = NULL; PosCorrente = -1; };
```

4.2.4.15 template<class T> int TAbstractList< T >::Search (T * *procurado*) [protected]

Busca um elemento na estrutura.

Busca o primeiro elemento da estrutura que seja igual a `procurado` e retorna sua posição.

Parâmetros:

procurado Elemento a ser buscado na estrutura

Retorna:

Posição do elemento, caso tenha encontrado algum, ou -1 caso o elemento não tenha sido encontrado.

Complexidade: $O(n)$

Reimplementado por TList< T >, TQueue< T >, TSortedList< T > e TStack< T >.

Definição na linha 927 do arquivo TAbstractList.h.

```
928 {
929         //Pocurando por u melemento nulo
930         if(procurado == NULL)
931                 throw new TENullElement("TAbstractList::Search");
932
933         //Varre todos os elementos
934         MoveFirst();
935         while( pCurrent != NULL )
936         {
937                 //Encontrou?
938                 if( *procurado == *(pCurrent->Data) ) { break; }
939
940                 //Proximo...
941                 MoveNext();
942         }
943
944         //Retorna o elemento encontrado
945         return PosCorrente;
946 }
```

4.2.4.16 template<class T> void TAbstractList< T >::Sort () [protected]

Ordena a estrutura em ordem crescente.

Esse método reseta o elemento corrente da estrutura.

Complexidade: $O(n \cdot \lg_2 n)$

Reimplementado por TList< T >.

Definição na linha 953 do arquivo TAbstractList.h.

Referenciado por TList< T >::Sort().

```
954 {
955         //Lista vazia
956         if(Size == 0)
957         {
```

```
958                 ResetCurrent();
959                 return;
960         }
961
962         //Array para ordenação
963         Node* Espelho[Size];
964         register int i;
965
966         //Armazena no array
967         for(i = MoveFirst(); i > -1; i = MoveNext())
968         {
969                 Espelho[i] = pCurrent;
970         }
971
972         //Ordena com o quicksort da bibliotecapadrao C
973         qsort(Espelho, Size, sizeof(Node *), _Compara<T>);
974
975         //Devolve pra lista
976         pCurrent = pFirst = Espelho[0];
977         pCurrent->pPrior = NULL;
978         for(i = 0; i < Size - 1; i++)   //Talvez esse moveNext()
979         {
980 std::cerr << " INICIO FOR ( " << i << ")" << std::endl;
981
982                 pCurrent->pNext = Espelho[i+1];
983                 pCurrent->pNext->pPrior = pCurrent;
984
985 std::cerr << " FIM FOR ( " << i << ")" << std::endl;
986                 pCurrent = pCurrent->pNext;
987         }
988
989         pLast = Espelho[Size-1];
990         pLast->pNext = NULL;
991
992 std::cout << " ORDENOU " << std::endl;
993
994         //Reseta o elemento corrente
995         ResetCurrent();
996 }
```

4.2.4.17 template<class T> Node* TAbstractList< T >::_CreateNode (T * *dado*, Node * *anterior*, Node * *posterior*) [inline, private]

Cria um novo elemento para a estrutura.

Função de apoio para tornar a programação ainda mais em alto nível.

Parâmetros:

dado Dado do elemento

anterior Apontador para o elemento anterior

posterior Apontador para o elemento posterior

Pós-Condição:

```
newNode->pNext = pCurrent
```

Complexidade: $O(1)$

Definição na linha 378 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_CreateNode() e TAbstractList< T >::Insert().

```
378 { Node *newNode = new Node; newNode->Data = dado; newNode->pPrior = anterior; newNode->pNext =
```

4.2.4.18 template<class T> bool TAbstractList< T >::_NodeSetPrior (Node * *node*, Node * *anterior*) [inline, private]

Define o pPrior de um nó (caso ele realmente seja um nó).

Aponta o elemento `anterior` como sendo o elemento anterior ao elemento `node`

Parâmetros:

node Elemento que terá seu antecessor alterado

anterior Elemento que será o antecessor

Retorna:

false caso `node` seja nulo, **true** caso contrário.

Pós-Condição:

```
Node* node->pPrior = (Node*) anterior;
```

(Se `node` for não-nulo)

Complexidade: $O(1)$

Definição na linha 398 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_NodeSetPrior() e TAbstractList< T >::_PrepareDel().

```
398 { if(node != NULL) { node->pPrior = anterior; return true; } else { return false; } };
```

4.2.4.19 template<class T> bool TAbstractList< T >::_NodeSetNext (Node * *node*, Node * *proximo*) [inline, private]

Define o pNext de um nó (caso ele realmente seja um nó).

Aponta o elemento `posterior` como sendo o elemento posteior ao elemento `node`

Parâmetros:

node Elemento que terá seu sucessor alterado

proximo Elemento que será o sucessor

Retorna:

false caso `node` seja nulo, **true** caso contrário.

Pós-Condição:

```
Node* node->pNext = (Node*) proximo;
```

(Se `node` for não-nulo)

Complexidade: $O(1)$

Definição na linha 418 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_NodeSetNext() e TAbstractList< T >::_PrepareDel().

```
418 { if(node != NULL) { node->pNext = proximo; return true; } else { return false; } };
```

4.2.4.20 template<typename T> bool TAbstractList< T >::_ToNext () [private]

Aponta para o próximo elemento e atualiza a posição.

Método utilizado apenas para tonar o codigo mais legível e reaproveitar código **Complexidade:** $O(1)$

Definição na linha 612 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::GoTo() e TAbstractList< T >::MoveNext().

```
613 {
614         //Retorna o status dessa ultima tentativa de movimentação
615
616         //Verifica se é possível mover p/ o proximo
617         if(pCurrent->pNext != NULL)
618         {
619                 //Move pro proximo
620                 PosCorrente++;
621                 pCurrent = pCurrent->pNext;
622
623                 return true;
624         }
625
626         return false;
627 }
```

4.2.4.21 template<typename T> bool TAbstractList< T >::_ToPrior () [private]

Aponta para o próximo elemento e atualiza a posição.

Método utilizado apenas para tonar o codigo mais legível e reaproveitar código **Complexidade:** $O(1)$

Definição na linha 634 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::GoTo() e TAbstractList< T >::MovePrior().

```
635 {
636         //Verifica se é possível mover p/ o anterior
637         if(pCurrent->pPrior != NULL)
638         {
639                 //Move pro anterior
640                 PosCorrente--;
641                 pCurrent = pCurrent->pPrior;
```

```
642
643                 return true;
644         }
645
646         return false;
647 }
```

4.2.4.22 template<class T> struct TAbstractList< T >::node * TAbstractList< T >::_PrepareDel (int *pos*) [private]

Prepara a lista para um Del.

Prepara a lista para que seja efetuado um Del e retorna o elemento da será o elemento corrente após a remoção. Método utilizado para tonar o codigo mais legível e reaproveitar código

Parâmetros:

pos Posição do elemento a ser removido

Pós-Condição:

```
pCurrent = GoTo(pos);
```

Complexidade: $O(n)$

Definição na linha 541 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::Del() e TAbstractList< T >::Kill().

```
542 {
543         //Fora dos limites
544         if( pos < 0 || pos >= Size )
545                 throw new TEOutOfBounds("TAbstractList::_PrepareDel");
546
547         //Armazenará o proximo pCurrent;
548         Node *pProximoCurrent = NULL;
549
550         //Move para a posição
551         GoTo(pos);
552
553         //
554         //Atualiza os ponteiros
555         //
556
557         //O elemento seguinte passa a ser o corrente [definição no trabalho]
558         pProximoCurrent = pCurrent->pNext;
559
560         //Liga o elemento anterior ao elemento posterior
561         //Próximo elemento do elemento anterior
562         //->
563         //Próximo elemento
564         if(! _NodeSetNext(pCurrent->pPrior, pCurrent->pNext) ) {        //Não existe anterior
565                 pFirst = pCurrent->pNext;                               //Cobre o caso em que
566                                                                         //Também cobre o caso
567         }
568
569         //Liga o elemento posterior ao anterior
570         //Elemento anterior do proximo elemento
```

```
571         //->
572         //Elemento anterior
573         if(! _NodeSetPrior(pCurrent->pNext, pCurrent->pPrior) ) {       //Não existe próximo
574                 PosCorrente = (Size > 1) ? PosCorrente - 1 : Size -1;   //Também cobre o caso d
575                 pProximoCurrent = pLast = pCurrent->pPrior;             //Também cobre o caso d
576         }
577
578         //Retorna o elemento que será o pCurrent
579         return pProximoCurrent;
580 }
```

4.2.4.23 template<class T> void TAbstractList< T >::_BubbleSort () [private]

Executa um Bubblesort na lista.

Complexidade: $O(n^2)$

Definição na linha 587 do arquivo TAbstractList.h.

```
588 {
589         //Otimização
590         register Node *i = NULL;
591         register Node *j = NULL;
592
593         //
594         //BubleSort utilizando ponteiros :-D
595         //Muito legal
596         //
597         for(i = pFirst; i != pLast->pPrior; i = i->pNext)
598                 for(j = pLast; j != i; j = j->pPrior)
599                         _Swap(j, j->pPrior);
600
601         //Reseta o elemento corrente
602         ResetCurrent();
603 }
```

4.2.4.24 template<class T> void TAbstractList< T >::_Swap (Node * *a*, Node * *b*) [inline, private]

Troca a posicao de 2 elementos.

Essa função é utilizada para auxiliar o _BubbleSort()

Parâmetros:

- ***a*** Primeiro elemento
- ***b*** Segundo elemento

Complexidade: $O(1)$

Definição na linha 492 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_BubbleSort() e TAbstractList< T >::_Swap().

```
492 { if ( *(a->Data) < *(b->Data) ) { T* aux = a->Data; a->Data = b->Data; b->Data = aux;} };
```

4.2.5 Campos e Atributos

4.2.5.1 template<class T> int TAbstractList< T >::Size [protected]

Número de elementos que a lista possui.

Definição na linha 89 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_PrepareDel(), TAbstractList< T >::Count(), TAbstractList< T >::Del(), TAbstractList< T >::GoTo(), TAbstractList< T >::Insert(), TAbstractList< T >::Kill(), TAbstractList< T >::MoveFirst(), TAbstractList< T >::MoveLast(), TAbstractList< T >::Sort() e TAbstractList< T >::TAbstractList().

4.2.5.2 template<class T> int TAbstractList< T >::PosCorrente [protected]

Posição do elemento apontado por TAbstractList::pFirst.

Definição na linha 90 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_PrepareDel(), TAbstractList< T >::_ToNext(), TAbstractList< T >::_ToPrior(), TAbstractList< T >::Del(), TAbstractList< T >::GoTo(), TAbstractList< T >::Insert(), TAbstractList< T >::Kill(), TAbstractList< T >::MoveFirst(), TAbstractList< T >::MoveLast(), TAbstractList< T >::MoveNext(), TAbstractList< T >::MovePrior(), TAbstractList< T >::ResetCurrent(), TAbstractList< T >::Search() e TAbstractList< T >::TAbstractList().

4.2.5.3 template<class T> Node* TAbstractList< T >::pFirst [protected]

Apontador para o primeiro elemento da lista.

Definição na linha 92 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_BubbleSort(), TAbstractList< T >::_PrepareDel(), TAbstractList< T >::Insert(), TAbstractList< T >::MoveFirst(), TAbstractList< T >::Sort() e TAbstractList< T >::TAbstractList().

4.2.5.4 template<class T> Node* TAbstractList< T >::pCurrent [protected]

Apontador para o elemento corrente da lista.

Veja também:
TAbstractList::PosCorrente

Definição na linha 93 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_PrepareDel(), TAbstractList< T >::_ToNext(), TAbstractList< T >::_ToPrior(), TAbstractList< T >::Del(), TAbstractList< T >::DelAll(), TAbstractList< T >::GetCurrent(), TAbstractList< T >::Insert(), TAbstractList< T >::Kill(), TAbstractList< T >::KillAll(), TAbstractList< T

>::MoveFirst(), TAbstractList< T >::MoveLast(), TAbstractList< T >::MoveNext(), TAbstractList< T >::MovePrior(), TAbstractList< T >::ResetCurrent(), TAbstractList< T >::Search(), TAbstractList< T >::Sort() e TAbstractList< T >::TAbstractList().

4.2.5.5 template<class T> Node* TAbstractList< T >::pLast [protected]

Apontador para o último elemento da lista.

Definição na linha 94 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_BubbleSort(), TAbstractList< T >::_PrepareDel(), TAbstractList< T >::Insert(), TAbstractList< T >::MoveLast(), TAbstractList< T >::Sort() e TAbstractList< T >::TAbstractList().

4.3 Referência da Estrutura TAbstractList< T >::node

4.3.1 Descrição Detalhada

template<class T> struct TAbstractList< T >::node

Um Nó que representa um elemento da lista.

Definição na linha 67 do arquivo TAbstractList.h.

Campos de Dados

- node * pPrior
  Elemento anterior ao nó.
- T * Data
  Dado que o nó contém.
- node * pNext
  Elemento posterior ao nó.

4.3.2 Campos e Atributos

4.3.2.1 template<class T> struct node* TAbstractList< T >::node::pPrior

Elemento anterior ao nó.

Definição na linha 68 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_BubbleSort(), TAbstractList< T >::_CreateNode(), TAbstractList< T >::_NodeSetPrior(), TAbstractList< T >::_PrepareDel(), TAbstractList< T >::_ToPrior(), TAbstractList< T >::Insert() e TAbstractList< T >::Sort().

4.3.2.2 template<class T> T* TAbstractList< T >::node::Data

Dado que o nó contém.

Definição na linha 69 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_CreateNode(), TAbstractList< T >::_Swap(), TAbstractList< T >::GetCurrent(), TAbstractList< T >::Kill() e TAbstractList< T >::Search().

4.3.2.3 template<class T> struct node* TAbstractList< T >::node::pNext

Elemento posterior ao nó.

Definição na linha 70 do arquivo TAbstractList.h.

Referenciado por TAbstractList< T >::_BubbleSort(), TAbstractList< T >::_CreateNode(), TAbstractList< T >::_NodeSetNext(), TAbstractList< T >::_PrepareDel(), TAbstractList< T >::_ToNext(), TAbstractList< T >::Insert() e TAbstractList< T >::Sort().

4.4 Referência da Template de Classe TApplication< T >

`#include <TApplication.h>`

4.4.1 Descrição Detalhada

template<class T> class TApplication< T >

Classe que implementa a aplicação central.

Definição na linha 37 do arquivo TApplication.h.

Métodos Públicos

- ~TApplication ()
  Destrutor.
- void Init (int argc, char *argv[ ])
  Inicializa a aplicação.
- void ShowUsage ()
  Exibe a tela de ajuda para a aplicação.
- bool ValidaTipoEstrutura ()
  Valida o tipo de estrutura.
- bool ValidaArquivoInput ()
  Valida o arquivo de INPUT.

- bool ValidaArquivoSearch ()
  Valida o arquivo de SEARCH.
- bool InitEngine ()
  Inicializa a engine especifica de cada estrutura de dados.
- bool ProcessaInput ()
  Processa o arquivo de INPUT.
- bool ProcessaSearch ()
  Processa o arquivo de SEARCH.

Atributos Protegidos

- TEngine< T > * engine
  Engine especifica da estrutura de dados selecionada.
- char iFile [255]
  Nome do arquivo de INPUT.
- char sFile [255]
  Nome do arquivo de SEARCH.
- char ** argumentos
  Lista de argumentos.
- FILE * input
  Descritor de Arquivo para iFile.
- FILE * search
  Descritor de Arquivo para sFile.
- tipo_estrutura tipo
  Indicador do tipo de estrutura que está sendo utilizada no programa.

Tipos Privados

- enum tipo_estrutura {
  L, SL, S, Q,
  H }
  Tipo de estrutura.

4.4.2 Enumerações

4.4.2.1 template<class T> enum TApplication::tipo_estrutura [private]

Tipo de estrutura.

Valores enumerados:

L Lista (TList).

SL Lista Ordenada (TSortedList).

S Pilha (TStack).

Q Fila (TQueue).

H Tabela Hash (THashTable).

Definição na linha 41 do arquivo TApplication.h.

```
42         {
43                 L,
44                 SL,
45                 S,
46                 Q,
47                 H
48         };
```

4.4.3 Construtores & Destrutores

4.4.3.1 template<class T> TApplication< T >::~TApplication ()

Destrutor.

Destrói o que deve ser destruído ao se terminar a aplicacao

Definição na linha 400 do arquivo TApplication.h.

```
401 {
402         //Fecha os arquivos
403         fclose(input);
404         fclose(search);
405
406         //Limpa a engine
407         engine->clean();
408 }
```

4.4.4 Métodos

4.4.4.1 template<class T> void TApplication< T >::Init (int *argc*, char * *argv*[ ])

Inicializa a aplicação.

Parâmetros:

argc Numero de argumentos

argv Lista de argumentos

Definição na linha 127 do arquivo TApplication.h.

```
128 {
129         //Aponta para os argumentos
130         argumentos = argv;
131
132         //Verifica se o programa foi chamado da maneira correta
133         if(argc < 4 || !strcmp(argv[1], "--help"))
134         {
135                 ShowUsage();
136                 exit(0);
137         }
138 }
```

4.4.4.2 template<class T> void TApplication< T >::ShowUsage ()

Exibe a tela de ajuda para a aplicação.

Definição na linha 142 do arquivo TApplication.h.

Referenciado por TApplication< T >::Init(), TApplication< T >::ValidaArquivoInput() e TApplication< T >::ValidaArquivoSearch().

```
143 {
144         std::cerr
145                 << "Uso: "<< argumentos[0] << " ESTRUTURA INPUT.txt BUSCA.txt" << std::endl <<
146                 << "ESTUTURA:" << std::endl
147                 << "\t -l \t TList" << std::endl
148                 << "\t -sl \t TSortedList" << std::endl
149                 << "\t -s \t TStack" << std::endl
150                 << "\t -q \t TQueue" << std::endl
151                 << "\t -h \t THash" << std::endl;
152 }
```

4.4.4.3 template<class T> bool TApplication< T >::ValidaTipoEstrutura ()

Valida o tipo de estrutura.

Valida o primeiro parâmetro passado ao programa e trata os possíveis erros

Definição na linha 156 do arquivo TApplication.h.

```
157 {
158         //Instancia o objeto
159         if(!strcmp(argumentos[1], "-l"))
160         {
161                 tipo = L;
162         }
163         else if(!strcmp(argumentos[1], "-sl"))
164         {
165                 tipo = SL;
166         }
167         else if(!strcmp(argumentos[1], "-s"))
168         {
169                 tipo = S;
170         }
171         else if(!strcmp(argumentos[1], "-q"))
```

```
172         {
173                 tipo = Q;
174         }
175         else if(!strcmp(argumentos[1], "-h"))
176         {
177                 tipo = H;
178         }
179         else
180         {
181                 const char *txt_erro = "ESTRUTURA inválida: ";
182                 int tamanho = strlen(txt_erro) + strlen(argumentos[1]);
183
184                 char msg_erro[++tamanho];
185
186                 TEGeneralError *ErrorHandler = new TEGeneralError(argumentos[0], strcat(strcpy(
187                 ErrorHandler->Exit(1);
188         }
189
190         return true;
191 }
```

4.4.4.4 template<class T> bool TApplication< T >::ValidaArquivoInput ()

Valida o arquivo de INPUT.

Valida o segundo parâmetro passado ao programa e trata os possíveis erros

Definição na linha 193 do arquivo TApplication.h.

```
194 {
195         //Abre o arquivo
196         if(strlen(argumentos[2]))
197         {
198                 if ( (input = fopen(argumentos[2], "r")) == NULL )
199                 {
200                         const char *txt_erro = "Erro ao abrir o arquivo INPUT: ";
201                         int tamanho = strlen(txt_erro) + strlen(argumentos[2]);
202
203                         char msg_erro[++tamanho];
204
205                         TEGeneralError *ErrorHandler = new TEGeneralError(argumentos[0], strcat
206                         ErrorHandler->Exit(2);
207
208                 }
209
210                 return true;
211         }
212         else
213         {
214                 ShowUsage();
215         }
216 }
```

4.4.4.5 template<class T> bool TApplication< T >::ValidaArquivoSearch ()

Valida o arquivo de SEARCH.

Valida o terceiro parâmetro passado ao programa e trata os possíveis erros

Definição na linha 220 do arquivo TApplication.h.

```
221 {
222         //Abre o arquivo
223         if(strlen(argumentos[3]))
224         {
225                 if ( (search = fopen(argumentos[3], "r")) == NULL )
226                 {
227                         const char *txt_erro = "Erro ao abrir o arquivo SEARCH: ";
228                         int tamanho = strlen(txt_erro) + strlen(argumentos[3]);
229
230                         char msg_erro[++tamanho];
231
232                         TEGeneralError *ErrorHandler = new TEGeneralError(argumentos[0], strcat
233                         ErrorHandler->Exit(3);
234                 }
235
236                 return true;
237         }
238         else
239         {
240                 ShowUsage();
241         }
242 }
```

4.4.4.6 template<class T> bool TApplication< T >::InitEngine ()

Inicializa a engine especifica de cada estrutura de dados.

Inicializa a parte específica do programa que tratará da estrutura de dados em questão

Definição na linha 246 do arquivo TApplication.h.

```
247 {
248         //Instancia a engine adequada
249         switch(tipo)
250         {
251                 //Lista
252                 case L:
253                   engine = new TEngineTList<T>(new TList<T>);
254                 break;
255
256                 //Lista Ordenada
257                 case SL:
258                   engine = new TEngineTSortedList<T>(new TSortedList<T>);
259                 break;
260
261                 //Pilha
262                 case S:
263                   engine = new TEngineTStack<T>(new TStack<T>);
264                 break;
265
266                 //Fila
267                 case Q:
268                   engine = new TEngineTQueue<T>(new TQueue<T>);
269                 break;
270
271                 //Tabela Hash
272                 case H:
273                   //Obtem o numero de elementos que a tabela armazenará
```

```
274                   //Gera um tamanho levemente menor que o numero de entradas, para fins demonst
275                   int tamanho;
276                   fscanf(input, "%d", &tamanho);
277                   tamanho -= (int) floor(log10(tamanho));
278
279                   engine = new TEngineTHashTable<T>(new THashTable<T>(tamanho));
280
281                   //reaponta o arquivo para o inicio
282                   rewind(input);
283                 break;
284
285                 //Nada (impossivel, mas ... )
286                 default:
287                   return false;
288                 break;
289         }
290 }
```

4.4.4.7 template<class T> bool TApplication< T >::ProcessaInput ()

Processa o arquivo de INPUT.

Processa o arquivo de INPUT e manipula as estruturas.

Definição na linha 294 do arquivo TApplication.h.

```
295 {
296         //Obtem o numero de comandos
297         int numRegistros;
298         fscanf(input, "%d", &numRegistros);
299
300 #ifdef DEBUG
301 std::cerr << numRegistros << " inputs " << endl;
302 #endif
303
304         //Processa o input
305         for(; numRegistros > 0; numRegistros--)
306         {
307                 //Cria o elemento
308                 T *elemento = new T();
309
310                 //Le a linha
311                 int posicao, codigo;
312                 char nome[255];
313                 fscanf(input, "%d\t%d\t%255[^\n]\n", &posicao, &codigo, &nome);
314
315                 //Preenche o elemento
316                 elemento->SetCodigo(codigo);
317                 elemento->SetNome(nome);
318
319 #ifdef DEBUG
320 std::cerr << "[pos " << posicao << "]\t[" << elemento << "]\t" << *elemento << std::endl;
321 #endif
322                 try {
323                         engine->InsereElemento(posicao, elemento);
324                 }
325                 catch(TEGeneralError *Error)
326                 {
327                         Error->toString();
328                 }
```

```
329         }
330
331
332 #ifdef DEBUG
333 std::cerr << std::endl << "Após as inserções: " << std::endl;
334 engine->dump();
335 #endif
336
337         return true;
338 }
```

4.4.4.8 template<class T> bool TApplication< T >::ProcessaSearch ()

Processa o arquivo de SEARCH.

Processa o arquivo de SEARCH e gera a saída do programa.

Definição na linha 340 do arquivo TApplication.h.

```
341 {
342         //Obtem o numero de comandos
343         int numOperacoes;
344         fscanf(search, "%d\n", &numOperacoes);
345
346         //Numero de intens nao encontrados
347         int naoEncontrados = 0;
348
349 #ifdef DEBUG
350 std::cerr << endl << numOperacoes << " searchs " << endl;
351 #endif
352
353         //Processa o input
354         for(; numOperacoes > 0; numOperacoes--)
355         {
356                 //Cria o elemento
357                 T *elemento = new T();
358
359                 //Le a linha
360                 int codigo;
361                 fscanf(search, "%d\n", &codigo);
362
363                 //Preenche o elemento
364                 elemento->SetCodigo(codigo);
365
366 #ifdef DEBUG
367 std::cerr << "[" << elemento << "]\t" << *elemento << std::endl;
368 #endif
369
370                 try {
371                         //Procura o elemento
372                         if (NULL != (elemento = engine->ProcuraElemento(elemento)) )
373                         {
374                                 //Estranho.. Utilizando a otimização (g++ -O3 [...] ) de repent
375                                 //Desabilitei a otimização e foi...
376
377                                 //Encontrou
378                                 std::cout << *elemento << std::endl;
379                         }
380                         else
381                         {
```

```
382                                 //Nao encontrou
383                                 std::cout << codigo << " - [Item not found]" << std::endl;
384                                 naoEncontrados++;
385                         }
386                 }
387                 catch(TEGeneralError *Error)
388                 {
389                         Error->toString();
390                 }
391         }
392
393         //Numero de itens nao encotrados
394         std::cout << "=> " << naoEncontrados << " item(s) not found." << std::endl;
395
396         return true;
397 }
```

4.4.5 Campos e Atributos

4.4.5.1 template<class T> TEngine<T>* TApplication< T >::engine [protected]

Engine especifica da estrutura de dados selecionada.

Definição na linha 51 do arquivo TApplication.h.

Referenciado por TApplication< T >::InitEngine(), TApplication< T >::ProcessaInput(), TApplication< T >::ProcessaSearch() e TApplication< T >::~TApplication().

4.4.5.2 template<class T> char TApplication< T >::iFile[255] [protected]

Nome do arquivo de INPUT.

Definição na linha 52 do arquivo TApplication.h.

4.4.5.3 template<class T> char TApplication< T >::sFile[255] [protected]

Nome do arquivo de SEARCH.

Definição na linha 53 do arquivo TApplication.h.

4.4.5.4 template<class T> char** TApplication< T >::argumentos [protected]

Lista de argumentos.

Definição na linha 54 do arquivo TApplication.h.

Referenciado por TApplication< T >::Init(), TApplication< T >::ShowUsage(), TApplication< T >::ValidaArquivoInput(), TApplication< T >::ValidaArquivo-Search() e TApplication< T >::ValidaTipoEstrutura().

4.4.5.5 template<class T> FILE* TApplication< T >::input [protected]

Descritor de Arquivo para iFile.

Definição na linha 55 do arquivo TApplication.h.

Referenciado por TApplication< T >::InitEngine(), TApplication< T >::ProcessaInput(), TApplication< T >::ValidaArquivoInput() e TApplication< T >::~TApplication().

4.4.5.6 template<class T> FILE* TApplication< T >::search [protected]

Descritor de Arquivo para sFile.

Definição na linha 56 do arquivo TApplication.h.

Referenciado por TApplication< T >::ProcessaSearch(), TApplication< T >::ValidaArquivoSearch() e TApplication< T >::~TApplication().

4.4.5.7 template<class T> tipo_estrutura TApplication< T >::tipo [protected]

Indicador do tipo de estrutura que está sendo utilizada no programa.

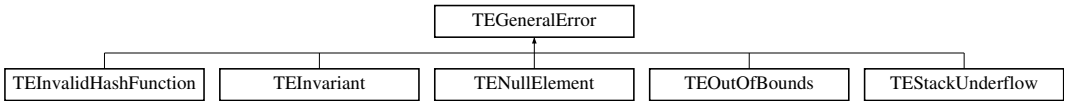
Definição na linha 57 do arquivo TApplication.h.

Referenciado por TApplication< T >::InitEngine() e TApplication< T >::ValidaTipoEstrutura().

4.5 Referência da Classe TEGeneralError

`#include <TError.h>`

Diagrama de Hierarquia para TEGeneralError::



4.5.1 Descrição Detalhada

Classe genérica de tratamento de erro. A classe TEGeneralError trata as exceções de maneira genérica, amazenando informações sobre a exceção em sí e as circunstâncias que a causaram.

Definição na linha 22 do arquivo TError.h.

Métodos Públicos

- TEGeneralError (const char *gerador, const char *mensagem)
  Construtor.
- ~TEGeneralError ()
  Destrutor.
- void toString ()
  Imprime o erro.
- void Exit (int cod)
  Finaliza o programa e exibe o erro.

Métodos Protegidos

- void SetGerador (const char *gerador)
  Define o método que gerou o erro.
- void SetMensagem (const char *mensagem)
  Define a mensagem de erro.

Atributos Protegidos

- const char * GeradorErro
  Nome do Método que gerou a exceção.
- const char * MensagemErro
  Mensagem de erro que descreve a exceção gerada.

4.5.2 Construtores & Destrutores

4.5.2.1 TEGeneralError::TEGeneralError (const char * *gerador*, const char * *mensagem*) [inline]

Construtor.

Parâmetros:

gerador Método que gerou a exceção

mensagem Mensagem de erro

Definição na linha 34 do arquivo TError.h.

```
34 : GeradorErro(gerador), MensagemErro(mensagem) {};
```

4.5.2.2 TEGeneralError::~TEGeneralError () [inline]

Destrutor.

Definição na linha 39 do arquivo TError.h.

```
39 { GeradorErro = NULL; MensagemErro = NULL;};
```

4.5.3 Métodos

4.5.3.1 void TEGeneralError::toString () [inline]

Imprime o erro.

Imprime o conteúdo do erro no formato: "[GeradorErro] ~> MensagemErro" para a saída padrão de erro

Definição na linha 46 do arquivo TError.h.

Referenciado por Exit().

```
46 {std::cerr << "[" << GeradorErro << "] ~> " << MensagemErro << std::endl;};
```

4.5.3.2 void TEGeneralError::Exit (int *cod*) [inline]

Finaliza o programa e exibe o erro.

Imprime o conteúdo do erro e aborta o programa com o código de saida `cod`

Parâmetros:

cod Código de saída do programa

Definição na linha 54 do arquivo TError.h.

Referenciado por Exit(), TApplication< T >::ValidaArquivoInput(), TApplication< T >::ValidaArquivoSearch() e TApplication< T >::ValidaTipoEstrutura().

```
54 { toString(); exit(cod); };
```

4.5.3.3 void TEGeneralError::SetGerador (const char * *gerador*) [inline, protected]

Define o método que gerou o erro.

Parâmetros:

gerador Método que gerou o erro

Veja também:

SetMensagem

Definição na linha 66 do arquivo TError.h.

Referenciado por SetGerador().

```
66 { GeradorErro = gerador; };
```

4.5.3.4 void TEGeneralError::SetMensagem (const char * *mensagem*) [inline, protected]

Define a mensagem de erro.

Parâmetros:

mensagem Mensagem de erros

Veja também:

SetGerador

Definição na linha 74 do arquivo TError.h.

Referenciado por SetMensagem().

```
74 { MensagemErro = mensagem; };
```

4.5.4 Campos e Atributos

4.5.4.1 const char* TEGeneralError::GeradorErro [protected]

Nome do Método que gerou a exceção.

Definição na linha 57 do arquivo TError.h.

Referenciado por SetGerador(), TEGeneralError(), toString() e ~TEGeneralError().

4.5.4.2 const char* TEGeneralError::MensagemErro [protected]

Mensagem de erro que descreve a exceção gerada.

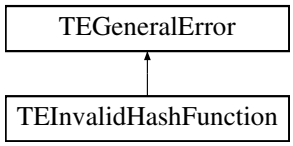
Definição na linha 58 do arquivo TError.h.

Referenciado por SetMensagem(), TEGeneralError(), toString() e ~TEGeneralError().

4.6 Referência da Classe TEInvalidHashFunction

`#include <TError.h>`

Diagrama de Hierarquia para TEInvalidHashFunction::



4.6.1 Descrição Detalhada

Exceção "Invalid Hash Function".

Indica que de alguam maneira a função que gera a chave Hash não funcionou como o esperado.

Definição na linha 124 do arquivo TError.h.

Métodos Públicos

- TEInvalidHashFunction (const char *gerador)
  Construtor.

4.6.2 Construtores & Destrutores

4.6.2.1 TEInvalidHashFunction::TEInvalidHashFunction (const char * *gerador*) [inline]

Construtor.

Parâmetros:
gerador Método que gerou a exceção

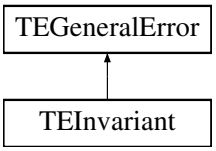
Definição na linha 130 do arquivo TError.h.

```
130 { SetGerador(gerador); SetMensagem("Invalid Hash Function"); };
```

4.7 Referência da Classe TEInvariant

`#include <TError.h>`

Diagrama de Hierarquia para TEInvariant::



4.7.1 Descrição Detalhada

Exceção "Invariant".

Indica que um Invariante foi violado. Utilizada apenas em tempo de desenvolvimento das classes

Definição na linha 138 do arquivo TError.h.

Métodos Públicos

- TEInvariant (const char *gerador)
  Construtor.

4.7.2 Construtores & Destrutores

4.7.2.1 TEInvariant::TEInvariant (const char * *gerador*) [inline]

Construtor.

Parâmetros:
gerador Método que gerou a exceção

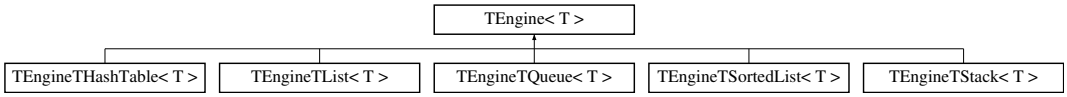
Definição na linha 144 do arquivo TError.h.

```
144 { SetGerador(gerador); SetMensagem("Invariant Error"); };
```

4.8 Referência da Template de Classe TEngine< T >

`#include <TEngine.h>`

Diagrama de Hierarquia para TEngine< T >::



4.8.1 Descrição Detalhada

template<class T> class TEngine< T >

Engine abstrata do programa.

Classe abstrata para definir a "engine" do programa principal facilitando a lógica do programa

Definição na linha 23 do arquivo TEngine.h.

Métodos Públicos

- virtual void clean ()=0
  Faz a limpeza dos objetos.

- virtual void InsereElemento (int posicao, T *elemento)=0
  Insere um elemento na estrutura.

- virtual T * ProcuraElemento (T *elemento)=0

  Procura um elemento na estrutura.

4.8.2 Métodos

4.8.2.1 template<class T> virtual void TEngine< T >::clean () [pure virtual]

Faz a limpeza dos objetos.

Tem o objetivo de funcionar como um destrutor

Implementado por TEngineTHashTable< T >, TEngineTList< T >, TEngineTQueue< T >, TEngineTSortedList< T > e TEngineTStack< T >.

4.8.2.2 template<class T> virtual void TEngine< T >::InsereElemento (int *posicao*, T * *elemento*) [pure virtual]

Insere um elemento na estrutura.

Insere o elemento `elemento` na posicao `posicao` Esse método é chamado a cada linha lida do arquivo de INPUT e é responsável pela inserção de cada elemento na estrutura em questão

Parâmetros:

posicao Posição onde o elemento será inserido

elemento O elemento que será inserido na lista

Implementado por TEngineTHashTable< T >, TEngineTList< T >, TEngineTQueue< T >, TEngineTSortedList< T > e TEngineTStack< T >.

4.8.2.3 template<class T> virtual T* TEngine< T >::ProcuraElemento (T * *elemento*) [pure virtual]

Procura um elemento na estrutura.

Retorna a posicao do primeiro elemento iagual a `elemento` existente na lista Esse método é chamado a cada linha lida do arquivo de SEARCH e é responsável pela busca de cada elemento na estrutura em questão

Parâmetros:

elemento O elemento que será procurado

Retorna:

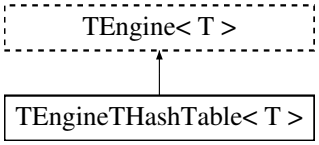
Ponteiro para o elemento encontrado

Implementado por TEngineTHashTable< T >, TEngineTList< T >, TEngineTQueue< T >, TEngineTSortedList< T > e TEngineTStack< T >.

4.9 Referência da Template de Classe TEngineTHashTable< T >

`#include <TEngineTHashTable.h>`

Diagrama de Hierarquia para TEngineTHashTable< T >::



4.9.1 Descrição Detalhada

template<class T> class TEngineTHashTable< T >

Engine referente à THashTable.

Essa engine trata o arquivo de INPUT e de SEARCH e manipula uma estrutura do tipo THashTable diretamente.

Definição na linha 23 do arquivo TEngineTHashTable.h.

Métodos Públicos

- TEngineTHashTable (THashTable< T > *t)
  Construtor.
- void clean ()
  Faz a limpeza dos objetos.
- void InsereElemento (int posicao, T *elemento)
  Insere um elemento na estrutura.
- T * ProcuraElemento (T *elemento)
  Procura um elemento na estrutura.

Campos de Dados

- THashTable< T > * TabelaHash
  Estrutura do tipo Lista que será utilizada para armazenar os elementos.

4.9.2 Construtores & Destrutores

4.9.2.1 template<class T> TEngineTHashTable< T >::TEngineTHashTable (THashTable< T > * *t*) [inline]

Construtor.

Parâmetros:

t Estrutura que será gerenciada pela engine

Definição na linha 31 do arquivo TEngineTHashTable.h.

```
31 : TabelaHash(t) { };
```

4.9.3 Métodos

4.9.3.1 template<class T> void TEngineTHashTable< T >::clean () [inline, virtual]

Faz a limpeza dos objetos.

Tem o objetivo de funcionar como um destrutor

Implementa TEngine< T >.

Definição na linha 39 do arquivo TEngineTHashTable.h.

```
39 { delete TabelaHash; };
```

4.9.3.2 template<class T> void TEngineTHashTable< T >::InsereElemento (int *posicao*, T * *elemento*) [inline, virtual]

Insere um elemento na estrutura.

Insere o elemento `elemento` na posicao `posicao` Esse método é chamado a cada linha lida do arquivo de INPUT e é responsável pela inserção de cada elemento na estrutura em questão

Parâmetros:

posicao Posição onde o elemento será inserido

elemento O elemento que será inserido na lista

Implementa TEngine< T >.

Definição na linha 40 do arquivo TEngineTHashTable.h.

Referenciado por TEngineTHashTable< T >::InsereElemento().

```
40 { TabelaHash->Add(elemento); };
```

4.9.3.3 template<class T> T* TEngineTHashTable< T >::ProcuraElemento (T * *elemento*) [inline, virtual]

Procura um elemento na estrutura.

Retorna a posicao do primeiro elemento iagual a `elemento` existente na lista Esse método é chamado a cada linha lida do arquivo de SEARCH e é responsável pela busca de cada elemento na estrutura em questão

Parâmetros:

elemento O elemento que será procurado

Retorna:

Ponteiro para o elemento encontrado

Implementa TEngine< T >.

Definição na linha 41 do arquivo TEngineTHashTable.h.

Referenciado por TEngineTHashTable< T >::ProcuraElemento().

```
41 { return TabelaHash->Get(elemento); };
```

4.9.4 Campos e Atributos

4.9.4.1 template<class T> THashTable<T>* TEngineTHashTable< T >::TabelaHash

Estrutura do tipo Lista que será utilizada para armazenar os elementos.

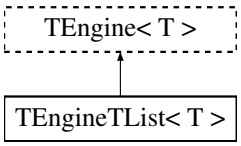
Definição na linha 25 do arquivo TEngineTHashTable.h.

Referenciado por TEngineTHashTable< T >::clean(), TEngineTHashTable< T >::InsereElemento(), TEngineTHashTable< T >::ProcuraElemento() e TEngineTHashTable< T >::TEngineTHashTable().

4.10 Referência da Template de Classe TEngineTList< T >

`#include <TEngineTList.h>`

Diagrama de Hierarquia para TEngineTList< T >::



4.10.1 Descrição Detalhada

template<class T> class TEngineTList< T >

Engine referente à TList.

Essa engine trata o arquivo de INPUT e de SEARCH e manipula uma estrutura do tipo TList diretamente.

Definição na linha 37 do arquivo TEngineTList.h.

Métodos Públicos

- TEngineTList (TList< T > *l)
  Construtor.
- void clean ()
  Faz a limpeza dos objetos.
- void InsereElemento (int posicao, T *elemento)
  Insere um elemento na estrutura.
- T * ProcuraElemento (T *elemento)
  Procura um elemento na estrutura.

Campos de Dados

- TList< T > * Lista
  Estrutura do tipo Lista que será utilizada para armazenar os elementos.

4.10.2 Construtores & Destrutores

4.10.2.1 template<class T> TEngineTList< T >::TEngineTList (TList< T > * *l*) [inline]

Construtor.

Parâmetros:

l Estrutura que será gerenciada pela engine

Definição na linha 45 do arquivo TEngineTList.h.

```
45 : Lista(l) { };
```

4.10.3 Métodos

4.10.3.1 template<class T> void TEngineTList< T >::clean () [inline, virtual]

Faz a limpeza dos objetos.

Tem o objetivo de funcionar como um destrutor

Implementa TEngine< T >.

Definição na linha 53 do arquivo TEngineTList.h.

```
53 { delete Lista; };
```

4.10.3.2 template<class T> void TEngineTList< T >::InsereElemento (int *posicao*, T * *elemento*) [inline, virtual]

Insere um elemento na estrutura.

Insere o elemento `elemento` na posicao `posicao` Esse método é chamado a cada linha lida do arquivo de INPUT e é responsável pela inserção de cada elemento na estrutura em questão

Parâmetros:

posicao Posição onde o elemento será inserido

elemento O elemento que será inserido na lista

Implementa TEngine< T >.

Definição na linha 54 do arquivo TEngineTList.h.

Referenciado por TEngineTList< T >::InsereElemento().

```
54 { Lista->Insert(elemento, posicao); } ;
```

4.10.3.3 template<class T> T* TEngineTList< T >::ProcuraElemento (T * *elemento*) [inline, virtual]

Procura um elemento na estrutura.

Retorna a posicao do primeiro elemento iagual a `elemento` existente na lista Esse método é chamado a cada linha lida do arquivo de SEARCH e é responsável pela busca de cada elemento na estrutura em questão

Parâmetros:

elemento O elemento que será procurado

Retorna:

Ponteiro para o elemento encontrado

Implementa TEngine< T >.

Definição na linha 55 do arquivo TEngineTList.h.

Referenciado por TEngineTList< T >::ProcuraElemento().

```
55 { int posicao; return (-1 != (posicao = Lista->Search(elemento)) ) ? Lista->Get(posicao) : NULL;
```

4.10.4 Campos e Atributos

4.10.4.1 template<class T> TList<T>* TEngineTList< T >::Lista

Estrutura do tipo Lista que será utilizada para armazenar os elementos.

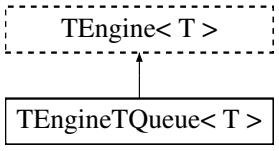
Definição na linha 39 do arquivo TEngineTList.h.

Referenciado por TEngineTList< T >::clean(), TEngineTList< T >::InsereElemento(), TEngineTList< T >::ProcuraElemento() e TEngineTList< T >::TEngineTList().

4.11 Referência da Template de Classe TEngineTQueue< T >

`#include <TEngineTQueue.h>`

Diagrama de Hierarquia para TEngineTQueue< T >::



4.11.1 Descrição Detalhada

template<class T> class TEngineTQueue< T >

Engine referente à TQueue.

Essa engine trata o arquivo de INPUT e de SEARCH e manipula uma estrutura do tipo TQueue diretamente.

Definição na linha 24 do arquivo TEngineTQueue.h.

Métodos Públicos

- TEngineTQueue (TQueue< T > *f)

  Construtor.

- void clean ()

  Faz a limpeza dos objetos.

- void InsereElemento (int posicao, T *elemento)
  Insere um elemento na estrutura.
- T * ProcuraElemento (T *elemento)
  Procura um elemento na estrutura.

Campos de Dados

- TQueue< T > * Fila
  Estrutura do tipo Pilha que será utilizada para armazenar os elementos.

4.11.2 Construtores & Destrutores

4.11.2.1 template<class T> TEngineTQueue< T >::TEngineTQueue (TQueue< T > * *f*) [inline]

Construtor.

Parâmetros:
f Estrutura que será gerenciada pela engine

Definição na linha 32 do arquivo TEngineTQueue.h.

```
32 : Fila(f) { };
```

4.11.3 Métodos

4.11.3.1 template<class T> void TEngineTQueue< T >::clean () [inline, virtual]

Faz a limpeza dos objetos.

Tem o objetivo de funcionar como um destrutor

Implementa TEngine< T >.

Definição na linha 39 do arquivo TEngineTQueue.h.

```
39 { delete Fila; };
```

4.11.3.2 template<class T> void TEngineTQueue< T >::InsereElemento (int *posicao*, T * *elemento*) [inline, virtual]

Insere um elemento na estrutura.

Insere o elemento `elemento` na posicao `posicao` Esse método é chamado a cada linha lida do arquivo de INPUT e é responsável pela inserção de cada elemento na estrutura em questão

Parâmetros:

posicao Posição onde o elemento será inserido

elemento O elemento que será inserido na lista

Implementa TEngine< T >.

Definição na linha 40 do arquivo TEngineTQueue.h.

Referenciado por TEngineTQueue< T >::InsereElemento().

```
40 { Fila->Append(elemento); };
```

4.11.3.3 template<class T> T* TEngineTQueue< T >::ProcuraElemento (T * *elemento*) [inline, virtual]

Procura um elemento na estrutura.

Retorna a posicao do primeiro elemento iagual a `elemento` existente na lista Esse método é chamado a cada linha lida do arquivo de SEARCH e é responsável pela busca de cada elemento na estrutura em questão

Parâmetros:

elemento O elemento que será procurado

Retorna:

Ponteiro para o elemento encontrado

Implementa TEngine< T >.

Definição na linha 41 do arquivo TEngineTQueue.h.

Referenciado por TEngineTQueue< T >::ProcuraElemento().

```
41 { int posicao; return (-1 != (posicao = Fila->Search(elemento)) ) ? Fila->Get(posicao) : NULL; }
```

4.11.4 Campos e Atributos

4.11.4.1 template<class T> TQueue<T>* TEngineTQueue< T >::Fila

Estrutura do tipo Pilha que será utilizada para armazenar os elementos.

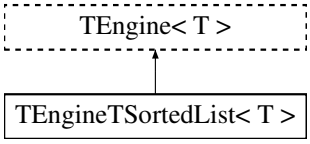
Definição na linha 26 do arquivo TEngineTQueue.h.

Referenciado por TEngineTQueue< T >::clean(), TEngineTQueue< T >::InsereElemento(), TEngineTQueue< T >::ProcuraElemento() e TEngineTQueue< T >::TEngineTQueue().

4.12 Referência da Template de Classe TEngineTSortedList< T >

`#include <TEngineTSortedList.h>`

Diagrama de Hierarquia para TEngineTSortedList< T >::



4.12.1 Descrição Detalhada

template<class T> class TEngineTSortedList< T >

Engine referente à TSortedList.

Essa engine trata o arquivo de INPUT e de SEARCH e manipula uma estrutura do tipo TSortedList diretamente.

Definição na linha 25 do arquivo TEngineTSortedList.h.

Métodos Públicos

- TEngineTSortedList (TSortedList< T > *l)
  Construtor.
- void clean ()
  Faz a limpeza dos objetos.
- void InsereElemento (int posicao, T *elemento)
  Insere um elemento na estrutura.
- T * ProcuraElemento (T *elemento)
  Procura um elemento na estrutura.

Campos de Dados

- TSortedList< T > * Lista
  Estrutura do tipo Lista Ordenada que será utilizada para armazenar os elementos.

4.12.2 Construtores & Destrutores

4.12.2.1 template<class T> TEngineTSortedList< T >::TEngineTSortedList (TSortedList< T > * *l*) [inline]

Construtor.

Parâmetros:

l Estrutura que será gerenciada pela engine

Definição na linha 33 do arquivo TEngineTSortedList.h.

```
33 : Lista(l) { };
```

4.12.3 Métodos

4.12.3.1 template<class T> void TEngineTSortedList< T >::clean () [inline, virtual]

Faz a limpeza dos objetos.

Tem o objetivo de funcionar como um destrutor

Implementa TEngine< T >.

Definição na linha 41 do arquivo TEngineTSortedList.h.

```
41 { delete Lista; };
```

4.12.3.2 template<class T> void TEngineTSortedList< T >::InsereElemento (int *posicao*, T * *elemento*) [inline, virtual]

Insere um elemento na estrutura.

Insere o elemento `elemento` na posicao `posicao` Esse método é chamado a cada linha lida do arquivo de INPUT e é responsável pela inserção de cada elemento na estrutura em questão

Parâmetros:

posicao Posição onde o elemento será inserido

elemento O elemento que será inserido na lista

Implementa TEngine< T >.

Definição na linha 42 do arquivo TEngineTSortedList.h.

Referenciado por TEngineTSortedList< T >::InsereElemento().

```
42 { Lista->Insert(elemento); };
```

4.12.3.3 template<class T> T* TEngineTSortedList< T >::ProcuraElemento (T * *elemento*) [inline, virtual]

Procura um elemento na estrutura.

Retorna a posicao do primeiro elemento iagual a `elemento` existente na lista Esse método é chamado a cada linha lida do arquivo de SEARCH e é responsável pela busca de cada elemento na estrutura em questão

Parâmetros:

elemento O elemento que será procurado

Retorna:
Ponteiro para o elemento encontrado

Implementa TEngine< T >.

Definição na linha 43 do arquivo TEngineTSortedList.h.

Referenciado por TEngineTSortedList< T >::ProcuraElemento().

```
43 { int posicao; return (-1 != (posicao = Lista->Search(elemento)) ) ? Lista->Get(posicao) : NULL;
```

4.12.4 Campos e Atributos

4.12.4.1 template<class T> TSortedList<T>* TEngineTSortedList< T >::Lista

Estrutura do tipo Lista Ordenada que será utilizada para armazenar os elementos.

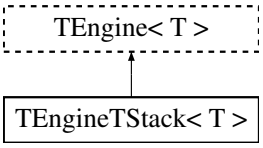
Definição na linha 27 do arquivo TEngineTSortedList.h.

Referenciado por TEngineTSortedList< T >::clean(), TEngineTSortedList< T >::InsereElemento(), TEngineTSortedList< T >::ProcuraElemento() e TEngineTSortedList< T >::TEngineTSortedList().

4.13 Referência da Template de Classe TEngineTStack< T >

`#include <TEngineTStack.h>`

Diagrama de Hierarquia para TEngineTStack< T >::



4.13.1 Descrição Detalhada

template<class T> class TEngineTStack< T >

Engine referente à TList.

Essa engine trata o arquivo de INPUT e de SEARCH e manipula uma estrutura do tipo TList diretamente.

Definição na linha 24 do arquivo TEngineTStack.h.

Métodos Públicos

- TEngineTStack (TStack< T > *p)

Construtor.

- void clean ()

  Faz a limpeza dos objetos.

- void InsereElemento (int posicao, T *elemento)

  Insere um elemento na estrutura.

- T * ProcuraElemento (T *elemento)

  Procura um elemento na estrutura.

Campos de Dados

- TStack< T > * Pilha

  Estrutura do tipo Pilha que será utilizada para armazenar os elementos.

4.13.2 Construtores & Destrutores

4.13.2.1 template<class T> TEngineTStack< T >::TEngineTStack (TStack< T > * *p*) [inline]

Construtor.

Parâmetros:

p Estrutura que será gerenciada pela engine

Definição na linha 32 do arquivo TEngineTStack.h.

```
32 : Pilha(p) { };
```

4.13.3 Métodos

4.13.3.1 template<class T> void TEngineTStack< T >::clean () [inline, virtual]

Faz a limpeza dos objetos.

Tem o objetivo de funcionar como um destrutor

Implementa TEngine< T >.

Definição na linha 40 do arquivo TEngineTStack.h.

```
40 { delete Pilha; };
```

4.13.3.2 template<class T> void TEngineTStack< T >::InsereElemento (int *posicao*, T * *elemento*) [inline, virtual]

Insere um elemento na estrutura.

Insere o elemento `elemento` na posicao `posicao` Esse método é chamado a cada linha lida do arquivo de INPUT e é responsável pela inserção de cada elemento na estrutura em questão

Parâmetros:

posicao Posição onde o elemento será inserido

elemento O elemento que será inserido na lista

Implementa TEngine< T >.

Definição na linha 41 do arquivo TEngineTStack.h.

Referenciado por TEngineTStack< T >::InsereElemento().

```
41 { Pilha->Push(elemento); };
```

4.13.3.3 template<class T> T* TEngineTStack< T >::ProcuraElemento (T * *elemento*) [inline, virtual]

Procura um elemento na estrutura.

Retorna a posicao do primeiro elemento iagual a `elemento` existente na lista Esse método é chamado a cada linha lida do arquivo de SEARCH e é responsável pela busca de cada elemento na estrutura em questão

Parâmetros:

elemento O elemento que será procurado

Retorna:

Ponteiro para o elemento encontrado

Implementa TEngine< T >.

Definição na linha 42 do arquivo TEngineTStack.h.

Referenciado por TEngineTStack< T >::ProcuraElemento().

```
42 { int posicao; return (-1 != (posicao = Pilha->Search(elemento)) ) ? Pilha->Get(posicao) : NULL;
```

4.13.4 Campos e Atributos

4.13.4.1 template<class T> TStack<T>* TEngineTStack< T >::Pilha

Estrutura do tipo Pilha que será utilizada para armazenar os elementos.

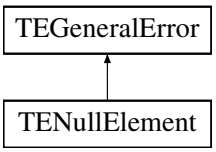
Definição na linha 26 do arquivo TEngineTStack.h.

Referenciado por TEngineTStack< T >::clean(), TEngineTStack< T >::InsereElemento(), TEngineTStack< T >::ProcuraElemento() e TEngineTStack< T >::TEngineTStack().

4.14 Referência da Classe TENullElement

`#include <TError.h>`

Diagrama de Hierarquia para TENullElement::



4.14.1 Descrição Detalhada

Exceção "Null Element".

Indica que uma ação tentou manipular um elemento nulo (que aponta para um ponteiro não definido).

Definição na linha 96 do arquivo TError.h.

Métodos Públicos

- TENullElement (const char *gerador)

  Construtor.

4.14.2 Construtores & Destrutores

4.14.2.1 TENullElement::TENullElement (const char * *gerador*) [inline]

Construtor.

Parâmetros:

gerador Método que gerou a exceção

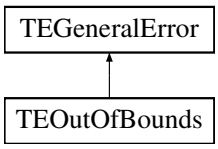
Definição na linha 102 do arquivo TError.h.

```
102 { SetGerador(gerador); SetMensagem("Tried to manipulate a NULL element"); };
```

4.15 Referência da Classe TEOutOfBounds

`#include <TError.h>`

Diagrama de Hierarquia para TEOutOfBounds::



4.15.1 Descrição Detalhada

Exceção "Out of Bounds".

Indica que uma ação excedeu os limites da lista.

Definição na linha 82 do arquivo TError.h.

Métodos Públicos

- TEOutOfBounds (const char *gerador)
  Construtor.

4.15.2 Construtores & Destrutores

4.15.2.1 TEOutOfBounds::TEOutOfBounds (const char * *gerador*) [inline]

Construtor.

Parâmetros:
gerador Método que gerou a exceção

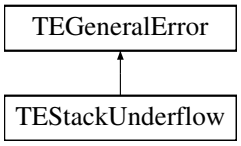
Definição na linha 88 do arquivo TError.h.

```
88 { SetGerador(gerador); SetMensagem("Index Out of Bounds"); };
```

4.16 Referência da Classe TEStackUnderflow

`#include <TError.h>`

Diagrama de Hierarquia para TEStackUnderflow::



4.16.1 Descrição Detalhada

Exceção "Stack Underflow".

Indica que a pilha sofreu um Underflow

Definição na linha 110 do arquivo TError.h.

Métodos Públicos

- TEStackUnderflow (const char *gerador)

  Construtor.

4.16.2 Construtores & Destrutores

4.16.2.1 TEStackUnderflow::TEStackUnderflow (const char * *gerador*) [inline]

Construtor.

Parâmetros:

gerador Método que gerou a exceção

Definição na linha 116 do arquivo TError.h.

```
116 { SetGerador(gerador); SetMensagem("Stack Underflow"); };
```

4.17 Referência da Template de Classe THashTable< T >

`#include <THashTable.h>`

4.17.1 Descrição Detalhada

template<class T> class THashTable< T >

Tabela Hash.

Versão:

1.0.1

Autor:

Reinaldo de Souza Junior <juniorz@gmail.com.br>

Data:

05/2005

Definição na linha 26 do arquivo THashTable.h.

Métodos Públicos

- THashTable ()
  Construtor padrão.
- THashTable (int tamanho, int(∗funcao)(T ∗, int)=NULL)
  Construtor.
- ∼THashTable ()
  Destrutor.
- int Add (T ∗data)
  Insere um elemento na tabela hash.
- T ∗ Get (T ∗data)
  Localiza um elemento na tabela e retorna-o.
- int Count ()
  Retorna o número de elementos da tabela hash.
- int Size ()
  Retorna o tamanho da tabela hash.
- bool Del (T ∗data)
  Remove um elemento da tabela hash.
- bool Kill (T ∗data)
  Remove e destrói um elemento da tabela hash.
- void DelAll ()
  Remove todos os elemento da tabela hash.
- void KillAll ()
  Remove todos os elemento da tabela hash.

Atributos Protegidos

- int iSize
  Tamanho da tabela hash.
- int iCount
  Número de elementos que a tabela possui.
- TList< T > ∗ Table
  Array de listas.

- int(∗ fHash)(T ∗, int)

  Função que calcula o hash.

Métodos Privados

- int Hash (T ∗data, int tamanho)

  Calcula o hash de um elemento.

4.17.2 Construtores & Destrutores

4.17.2.1 template<class T> THashTable< T >::THashTable () [inline]

Construtor padrão.

Fornecido para permitir que classes sejam implementadas a partir de THashTable

Complexidade: $O(1)$

Definição na linha 36 do arquivo THashTable.h.

```
36 : fHash(NULL) {};
```

4.17.2.2 template<class T> THashTable< T >::THashTable (int *tamanho*, int(∗ *funcao*)(T ∗, int) = NULL)

Construtor.

Parâmetros:

tamanho Tamanho da tabela hash

funcao Função que gera o hash

Complexidade: $O(1)$

Definição na linha 222 do arquivo THashTable.h.

```
222                                                                          : iSize(tamanho)
223 {
224         //Aloca espaço para a tabela
225         //Não precisa iniciar cada Lista, pois o construtor padrão é chamado automaticamente.
226         Table = new TList<T>[iSize];
227
228         /*
229          * Preferi não gerar a exceção e caso a funcao não seja informada,
230          * utilizar a função sugerida pelo professor
231          */
232 }
```

4.17.2.3 template<class T> THashTable< T >::~THashTable ()

Destrutor.

Complexidade: $O(1)$

Definição na linha 214 do arquivo THashTable.h.

```
215 {
216         //Limpa a memoria
217         //Isso deleta todos os elementos e chama o destrutor padrão de todos automaticamente
218         delete[] Table;
219 }
```

4.17.3 Métodos

4.17.3.1 template<class T> int THashTable< T >::Add (T * *data*)

Insere um elemento na tabela hash.

Parâmetros:

data Dado a ser inserido na tabela hash

Retorna:

Posição na tabela hash onde foi inserido o elemento

Exceções:

TENullElement*∗* caso `data` seja NULL. **Veja:** TENullElement

TEInvalidHashFunction*∗* caso o hash gerado seja inválido. **Veja:** TEInvalid-HashFunction

Complexidade: $O(1)$

Definição na linha 244 do arquivo THashTable.h.

```
245 {
246         //Elemento nulo
247         if(data == NULL)
248                 throw new TENullElement("THashTable::Add");
249 
250         /*
251          * Chave do item
252          * Utiliza por padrão a função fornecida pela classe
253          */
254         int chave = (NULL != fHash) ? fHash(data, iSize) : Hash(data, iSize);
255 
256 #ifdef DEBUG
257 std::cerr << "[chave] " << chave << std::endl;
258 #endif
259 
260         //Função inválida
261         if( chave < 0 || chave >= iSize )
262                 throw new TEInvalidHashFunction("THashTable::Add");
263 
264         //Insere o elemento
265         Table[chave].Insert(data, 0);
```

```
266
267         //Contador
268         iCount++;
269
270         //Retorna a posicao da tabela onde o elemento foi inserido
271         return chave;
272 }
```

4.17.3.2 template<class T> T * THashTable< T >::Get (T * *data*)

Localiza um elemento na tabela e retorna-o.

Parâmetros:

data Dado a ser procurado

Retorna:

Elemento caso seja encontrado, ou NULL caso contrário.

Exceções:

TENullElement*** caso `data` seja NULL. **Veja:** TENullElement

Complexidade: $O(1)$

Definição na linha 275 do arquivo THashTable.h.

```
276 {
277         //Elemento nulo
278         if(data == NULL)
279                 throw new TENullElement("THashTable::Get");
280
281         /*
282          * Chave do item
283          * Utiliza por padrão a função fornecida pela classe
284          */
285         int chave = (NULL != fHash) ? fHash(data, iSize) : Hash(data, iSize);
286
287         //Posicao do elemento na lista
288 //      int posicao = Table[chave].Search(data);
289
290         //Aproveita-se da exceção que será gerada ao passar uma posição inválida para o método
291         //caso o elemento não seja encontrado
292         try {
293                 return Table[chave].Get( Table[chave].Search(data) );
294         }
295         catch(TEOutOfBounds*)
296         {
297                 return NULL;
298         }
299 }
```

4.17.3.3 template<class T> int THashTable< T >::Count () `[inline]`

Retorna o número de elementos da tabela hash.

Retorna:
Número de elementos armazenados na tabela hash

Complexidade: $O(1)$

Definição na linha 98 do arquivo THashTable.h.

```
98 { return Count; };
```

4.17.3.4 template<class T> int THashTable< T >::Size () [inline]

Retorna o tamanho da tabela hash.

Retorna:
Tamanho da tabela

Complexidade: $O(1)$

Definição na linha 111 do arquivo THashTable.h.

```
111 { return Size; };
```

4.17.3.5 template<class T> bool THashTable< T >::Del (T * *data*)

Remove um elemento da tabela hash.

Remove o elemento da tabela hash que corresponda a `data`. Esse método não destrói o elemento apontado pelo nó, apenas retira-o da estrutura, para destruí-lo use Kill()

Parâmetros:
data Elemento a ser removido da lista.

Retorna:
True caso o elemento exista na tabela hash, False caso contrário.

Complexidade: $O(1)$

Definição na linha 301 do arquivo THashTable.h.

```
302 {
303         //Elemento nulo
304         if(data == NULL)
305                 throw new TENullElement("THashTable::Del");
306
307         /*
308          * Chave do item
309          * Utiliza por padrão a função fornecida pela classe
310          */
311         int chave = (NULL != fHash) ? fHash(data, iSize) : Hash(data, iSize);
312
313         //Aproveita-se da exceção que será gerada ao passar uma posição inválida para o método
314         //caso o elemento não seja encontrado
```

```
315         try {
316                 Table[chave].Del( Table[chave].Search(data) );
317                 iCount--;
318                 return true;
319         }
320         catch(TEOutOfBounds*)
321         {
322                 return false;
323         }
324 }
```

4.17.3.6 template<class T> bool THashTable< T >::Kill (T * *data*)

Remove e destrói um elemento da tabela hash.

Libera a memória ocupada pelo elemento que corresponde a `data` e retira-o da estrutura. Após a chamada desse método o elemento, qualquer referência ao elemento destruído será inválida.

Parâmetros:

data Elemento a ser destruído.

Retorna:

True caso o elemento exista na tabela hash, False caso contrário.

Complexidade: $O(1)$

Definição na linha 327 do arquivo THashTable.h.

```
328 {
329         //Elemento nulo
330         if(data == NULL)
331                 throw new TENullElement("THashTable::Kill");
332
333         /*
334          * Chave do item
335          * Utiliza por padrão a função fornecida pela classe
336          */
337         int chave = (NULL != fHash) ? fHash(data, iSize) : Hash(data, iSize);
338
339         //Aproveita-se da exceção que será gerada ao passar uma posição inválida para o método
340         //caso o elemento não seja encontrado
341         try {
342                 Table[chave].Kill( Table[chave].Search(data) ); //O(1)
343                 iCount--;
344                 return true;
345         }
346         catch(TEOutOfBounds*)
347         {
348                 return false;
349         }
350 }
```

4.17.3.7 template<class T> void THashTable< T >::DelAll ()

Remove todos os elemento da tabela hash.

Complexidade: $O(n)$

Definição na linha 353 do arquivo THashTable.h.

```
354 {
355         //Varre a tabela
356         int i;
357         for(i = 0; i < iSize; i++)      //O(n)
358         {
359                 Table[i].DelAll();      //O(1)
360         }
361 }
```

4.17.3.8 template<class T> void THashTable< T >::KillAll ()

Remove todos os elemento da tabela hash.

Complexidade: $O(n)$

Definição na linha 364 do arquivo THashTable.h.

```
365 {
366         //Varre a tabela
367         int i;
368         for(i = 0; i < iSize; i++)      //O(n)
369         {
370                 Table[i].KillAll();     //O(1)
371         }
372 }
```

4.17.3.9 template<class T> int THashTable< T >::Hash (T * *data*, int *tamanho*) [private]

Calcula o hash de um elemento.

Parâmetros:

data Dado a será manipulado

tamanho Tamanho da tabela hash

Retorna:

Posição na tabela hash onde o elemento será inserido

Complexidade: $O(1)$

Definição na linha 235 do arquivo THashTable.h.

Referenciado por THashTable< T >::Add(), THashTable< T >::Del(), THashTable< T >::Get() e THashTable< T >::Kill().

```
236 {
237         //Constante mágica sugerida pelo Sr. Donald E. Knuth
238         float k = (sqrt(5) - 1) / 2;
239
240         return (int) floor( tamanho * ( (int) (*data) * k - floor( (int) (*data) * k ) ) );
241 }
```

4.17.4 Campos e Atributos

4.17.4.1 template<class T> int THashTable< T >::iSize [protected]

Tamanho da tabela hash.

Definição na linha 190 do arquivo THashTable.h.

Referenciado por THashTable< T >::Add(), THashTable< T >::Del(), THashTable< T >::DelAll(), THashTable< T >::Get(), THashTable< T >::Kill(), THashTable< T >::KillAll() e THashTable< T >::THashTable().

4.17.4.2 template<class T> int THashTable< T >::iCount [protected]

Número de elementos que a tabela possui.

Definição na linha 191 do arquivo THashTable.h.

Referenciado por THashTable< T >::Add(), THashTable< T >::Del() e THashTable< T >::Kill().

4.17.4.3 template<class T> TList<T>* THashTable< T >::Table [protected]

Array de listas.

É o coração da tabela Hash

Definição na linha 192 do arquivo THashTable.h.

Referenciado por THashTable< T >::Add(), THashTable< T >::Del(), THashTable< T >::DelAll(), THashTable< T >::Get(), THashTable< T >::Kill(), THashTable< T >::KillAll(), THashTable< T >::THashTable() e THashTable< T >::~THashTable().

4.17.4.4 template<class T> int(* THashTable< T >::fHash)(T *, int) [protected]

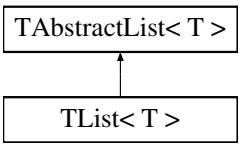
Função que calcula o hash.

Referenciado por THashTable< T >::Add(), THashTable< T >::Del(), THashTable< T >::Get(), THashTable< T >::Kill() e THashTable< T >::THashTable().

4.18 Referência da Template de Classe TList< T >

`#include <TList.h>`

Diagrama de Hierarquia para TList< T >::



4.18.1 Descrição Detalhada

template<class T> class TList< T >

Lista.

Estrutura de Lista

Versão:
1.0

Autor:
Reinaldo de Souza Junior <juniorz@gmail.com.br>

Data:
05/2005

Definição na linha 25 do arquivo TList.h.

Métodos Públicos

- int Count ()
  Retorna o numero de elementos da estrutura.
- T * Get (int pos)
  Retorna um elemento de uma posição específica.
- T * GetCurrent ()
  Retorna o elemento corrente da estrutura.
- int Insert (T *data, int pos)
  Insere um elemento em uma posição específica.
- int Del (int pos)
  Remove um elemento da estrutura.
- void DelAll ()
  Remove todos os elemento da estrutura.
- int Kill (int pos)
  Remove e destrói um elemento da estrutura.
- void KillAll ()
  Remove e destrói todos os elemento da estrutura.
- int MoveFirst ()
  Aponta o elemento corrente para o primeiro elemento da estrutura.

- int MoveNext ()

  Aponta o elemento corrente para próximo elemento da estrutura.

- int MovePrior ()

  Aponta o elemento corrente para o elemento anterior da estrutura.

- int MoveLast ()

  Aponta o elemento corrente para o último elemento da estrutura.

- int GoTo (int pos)

  Aponta o elemento corrente para o elemento na posição `pos`.

- void ResetCurrent ()

  Reseta o elemento corrente da estrutura.

- int Search (T *procurado)

  Busca um elemento na estrutura.

- void Sort ()

  Ordena a estrutura em ordem crescente.

4.18.2 Métodos

4.18.2.1 template<class T> int TList< T >::Count () `[inline]`

Retorna o numero de elementos da estrutura.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 32 do arquivo TList.h.

```
32 { return TAbstractList<T>::Count(); };
```

4.18.2.2 template<class T> T* TList< T >::Get (int *pos*) `[inline]`

Retorna um elemento de uma posição específica.

Retorna um ponteiro para elemento da estrutura que ocupa a posição `pos`

Parâmetros:

pos Posição do elemento que se deseja obter

Retorna:

Ponteiro associado ao nó que ocupa a posição `pos` na estrutura

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 39 do arquivo TList.h.

Referenciado por TList< T >::Get().

```
39 { return TAbstractList<T>::Get(pos); };
```

4.18.2.3 template<class T> T* TList< T >::GetCurrent () [inline]

Retorna o elemento corrente da estrutura.

Retorna:
Ponteiro associado ao nó que ocupa a posição corrente na estrutura.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 46 do arquivo TList.h.

```
46 { return TAbstractList<T>::GetCurrent(); };
```

4.18.2.4 template<class T> int TList< T >::Insert (T * *data*, int *pos*) [inline]

Insere um elemento em uma posição específica.

Insere um elemento que aponta para `data` na posição `pos` da estrutura.

Parâmetros:
data Ponteiro para um elemento que será inserido na estrutura
pos Posição onde o elemento deve ser inserido

Retorna:
Posição onde o elemento foi inserido

Exceções:
TENullElement caso `data` seja nulo (não aponta para nada). **Veja:** TENullElement
TEOutOfBounds caso `pos` seja uma posição inválida. **Veja:** TEOutOfBounds

Pós-Condição:
`data == Get(pos);`

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 53 do arquivo TList.h.

Referenciado por TList< T >::Insert().

```
53 { return TAbstractList<T>::Insert(data, pos); };
```

4.18.2.5 template<class T> int TList< T >::Del (int *pos*) [inline]

Remove um elemento da estrutura.

Remove o elemento da estrutura na posição `pos`. Esse método não destrói o elemento apontado pelo nó, apenas retira-o da estrutura, para destruí-lo use Kill()

Parâmetros:

pos Posição do elemento a ser removido da lista.

Retorna:

Posição do elemento corrente da estrutura após a remoção.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 60 do arquivo TList.h.

Referenciado por TList< T >::Del().

```
60 { return TAbstractList<T>::Del(pos); };
```

4.18.2.6 template<class T> void TList< T >::DelAll () [inline]

Remove todos os elemento da estrutura.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 67 do arquivo TList.h.

```
67 { TAbstractList<T>::DelAll(); };
```

4.18.2.7 template<class T> int TList< T >::Kill (int *pos*) [inline]

Remove e destrói um elemento da estrutura.

Libera a memória ocupada pelo elemento na posição `pos` e retira-o da estrutura. Após a chamada desse método o elemento, qualquer referência ao elemento destruído será inválida.

Parâmetros:

pos Posição do elemento a ser destruído.

Retorna:

Posição do elemento corrente da estrutura.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 74 do arquivo TList.h.

Referenciado por TList< T >::Kill().

```
74 { return TAbstractList<T>::Kill(pos); };
```

4.18.2.8 template<class T> void TList< T >::KillAll () [inline]

Remove e destrói todos os elemento da estrutura.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 81 do arquivo TList.h.

```
81 { TAbstractList<T>::KillAll(); };
```

4.18.2.9 template<class T> int TList< T >::MoveFirst () [inline]

Aponta o elemento corrente para o primeiro elemento da estrutura.

Retorna:
Posição do elemento corrente da estrutura (usualmente retorna 0)

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 88 do arquivo TList.h.

```
88 { return TAbstractList<T>::MoveFirst(); };
```

4.18.2.10 template<class T> int TList< T >::MoveNext () [inline]

Aponta o elemento corrente para próximo elemento da estrutura.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 95 do arquivo TList.h.

```
95 { return TAbstractList<T>::MoveNext(); };
```

4.18.2.11 template<class T> int TList< T >::MovePrior () [inline]

Aponta o elemento corrente para o elemento anterior da estrutura.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 102 do arquivo TList.h.

```
102 { return TAbstractList<T>::MovePrior(); };
```

4.18.2.12 template<class T> int TList< T >::MoveLast () [inline]

Aponta o elemento corrente para o último elemento da estrutura.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 109 do arquivo TList.h.

```
109 { return TAbstractList<T>::MoveLast(); };
```

4.18.2.13 template<class T> int TList< T >::GoTo (int *pos*) [inline]

Aponta o elemento corrente para o elemento na posição `pos`.

Parâmetros:
pos Posição do novo elemento corrente

Retorna:
Posição do novo elemento corrente

Pós-Condição:
PosCorrente = `pos`

Exceções:
TEOutOfBounds$*$ caso `pos` seja uma posição inválida

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 116 do arquivo TList.h.

Referenciado por TList< T >::GoTo().

```
116 { return TAbstractList<T>::GoTo(pos); };
```

4.18.2.14 template<class T> void TList< T >::ResetCurrent () [inline]

Reseta o elemento corrente da estrutura.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 123 do arquivo TList.h.

```
123 { TAbstractList<T>::ResetCurrent(); };
```

4.18.2.15 template<class T> int TList< T >::Search (T * *procurado*) [inline]

Busca um elemento na estrutura.

Busca o primeiro elemento da estrutura que seja igual a `procurado` e retorna sua posição.

Parâmetros:

procurado Elemento a ser buscado na estrutura

Retorna:

Posição do elemento, caso tenha encontrado algum, ou -1 caso o elemento não tenha sido encontrado.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 130 do arquivo TList.h.

Referenciado por TList< T >::Search().

```
130 { return TAbstractList<T>::Search(procurado); };
```

4.18.2.16 template<class T> void TList< T >::Sort () [inline]

Ordena a estrutura em ordem crescente.

Esse método reseta o elemento corrente da estrutura.

Complexidade: $O(n \cdot \lg_2 n)$

Reimplementação de TAbstractList< T >.

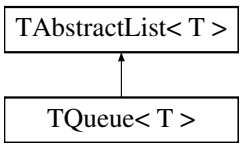
Definição na linha 137 do arquivo TList.h.

```
137 { TAbstractList<T>::Sort(); };
```

4.19 Referência da Template de Classe TQueue< T >

`#include <TQueue.h>`

Diagrama de Hierarquia para TQueue< T >::



4.19.1 Descrição Detalhada

template<class T> class TQueue< T >

Fila.

Estrutura de Fila

Versão:
1.0

Autor:
Reinaldo de Souza Junior <juniorz@gmail.com.br>

Data:
05/2005

Definição na linha 26 do arquivo TQueue.h.

Métodos Públicos

- int Append (T *data)
  Insere um elemento no final da fila.
- T * Get (int pos)
  Retorna um elemento de uma posição específica.
- T * GetCurrent ()
  Retorna o elemento corrente da lista.
- int Count ()
  Retorna o numero de elementos da pilha.
- int MoveFirst ()
  Aponta o elemento corrente para o primeiro elemento da lista.

- int MoveNext ()
  Aponta o elemento corrente para próximo elemento da lista.
- int MovePrior ()
  Aponta o elemento corrente para o elemento anterior da lista.
- int MoveLast ()
  Aponta o elemento corrente para o último elemento da lista.
- void ResetCurrent ()
  Reseta o elemento corrente da fila.
- int Search (T *procurado)
  Busca um elemento na fila.
- int Del ()
  Remove o primeiro elemento da fila.
- void DelAll ()
  Remove todos os elemento da pilha.
- int Kill ()
  Destrói o primeiro elemento da fila.
- void KillAll ()
  Destrói todos os elemento da pilha.

4.19.2 Métodos

4.19.2.1 template<class T> int TQueue< T >::Append (T * *data*) [inline]

Insere um elemento no final da fila.

Parâmetros:
data Ponteiro para um elemento que será inserido na fila

Retorna:
Posição do elemento inserido na fila

Exceções:
TENullElement caso `data` seja nulo (não aponta para nada). **Veja:** TENullElement

Veja também:
TEGeneralError

Definição na linha 42 do arquivo TQueue.h.

Referenciado por TQueue< T >::Append().

```
42 { return TAbstractList<T>::Insert(data, Count()); };
```

4.19.2.2 template<class T> T* TQueue< T >::Get (int *pos*) [inline]

Retorna um elemento de uma posição específica.

Retorna um ponteiro para elemento da lista que ocupa a posição `pos`

Parâmetros:

pos Posição do elemento que se deseja obter

Retorna:

Ponteiro associado ao nó que ocupa a posição `pos` na lista.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 59 do arquivo TQueue.h.

Referenciado por TQueue< T >::Get().

```
59 { return TAbstractList<T>::Get(pos); };
```

4.19.2.3 template<class T> T* TQueue< T >::GetCurrent () [inline]

Retorna o elemento corrente da lista.

Retorna:

Ponteiro associado ao nó que ocupa a posição corrente na lista.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 72 do arquivo TQueue.h.

```
72 { return TAbstractList<T>::GetCurrent(); };
```

4.19.2.4 template<class T> int TQueue< T >::Count () [inline]

Retorna o numero de elementos da pilha.

Veja também:

TAbstractList::Size

Reimplementação de TAbstractList< T >.

Definição na linha 83 do arquivo TQueue.h.

Referenciado por TQueue< T >::Append().

```
83 { return TAbstractList<T>::Count(); };
```

4.19.2.5 template<class T> int TQueue< T >::MoveFirst () [inline]

Aponta o elemento corrente para o primeiro elemento da lista.

Retorna:
Posição do elemento corrente da lista (usualmente retorna 0)

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 97 do arquivo TQueue.h.

```
97 { return TAbstractList<T>::MoveFirst(); };
```

4.19.2.6 template<class T> int TQueue< T >::MoveNext () [inline]

Aponta o elemento corrente para próximo elemento da lista.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 109 do arquivo TQueue.h.

```
109 { return TAbstractList<T>::MoveNext(); };
```

4.19.2.7 template<class T> int TQueue< T >::MovePrior () [inline]

Aponta o elemento corrente para o elemento anterior da lista.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 121 do arquivo TQueue.h.

```
121 { return TAbstractList<T>::MovePrior(); };
```

4.19.2.8 template<class T> int TQueue< T >::MoveLast () [inline]

Aponta o elemento corrente para o último elemento da lista.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 133 do arquivo TQueue.h.

```
133 { return TAbstractList<T>::MoveLast(); };
```

4.19.2.9 template<class T> void TQueue< T >::ResetCurrent () [inline]

Reseta o elemento corrente da fila.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 145 do arquivo TQueue.h.

```
145 { TAbstractList<T>::ResetCurrent(); };
```

4.19.2.10 template<class T> int TQueue< T >::Search (T * *procurado*) [inline]

Busca um elemento na fila.

Busca o primeiro elemento da fila que seja igual a "procurado" e retorna sua posição.

Parâmetros:

procurado Elemento a ser buscado na fila

Retorna:

Posição do elemento, caso tenha encontrado algum, ou -1 caso o elemento não tenha sido encontrado.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 162 do arquivo TQueue.h.

Referenciado por TQueue< T >::Search().

```
162 { return TAbstractList<T>::Search(procurado); } ;
```

4.19.2.11 template<class T> int TQueue< T >::Del () [inline]

Remove o primeiro elemento da fila.

Retorna:
Posição do elemento corrente da fila após a remoção.

Complexidade: $O(n)$

Definição na linha 176 do arquivo TQueue.h.

```
176 { return TAbstractList<T>::Del(0); };
```

4.19.2.12 template<class T> void TQueue< T >::DelAll () [inline]

Remove todos os elemento da pilha.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 188 do arquivo TQueue.h.

```
188 { TAbstractList<T>::DelAll(); };
```

4.19.2.13 template<class T> int TQueue< T >::Kill () [inline]

Destrói o primeiro elemento da fila.

Libera a memória ocupada pelo primeiro elemento da fila e retira-o. Após a chamada desse método o elemento, qualquer referência ao elemento destruído será inválida.

Retorna:
Posição do elemento corrente da fila.

Complexidade: $O(n)$

Definição na linha 205 do arquivo TQueue.h.

```
205 { return TAbstractList<T>::Kill(0); };
```

4.19.2.14 template<class T> void TQueue< T >::KillAll () [inline]

Destrói todos os elemento da pilha.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

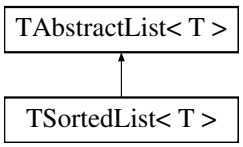
Definição na linha 217 do arquivo TQueue.h.

```
217 { TAbstractList<T>::KillAll(); };
```

4.20 Referência da Template de Classe TSortedList< T >

`#include <TSortedList.h>`

Diagrama de Hierarquia para TSortedList< T >::



4.20.1 Descrição Detalhada

template<class T> class TSortedList< T >

Lista Ordenada.

Estrutura de Lista Ordenada

Versão:
1.0

Autor:
Reinaldo de Souza Junior <juniorz@gmail.com.br>

Data:
05/2005

Definição na linha 25 do arquivo TSortedList.h.

Métodos Públicos

- int Count ()
  Retorna o numero de elementos da estrutura.
- T * Get (int pos)
  Retorna um elemento de uma posição específica.
- T * GetCurrent ()
  Retorna o elemento corrente da estrutura.
- int Insert (T *data)
  Insere um elemento ordenadamente.
- int Del (int pos)
  Remove um elemento da estrutura.

- void DelAll ()

  Remove todos os elemento da estrutura.

- int Kill (int pos)

  Remove e destrói um elemento da estrutura.

- void KillAll ()

  Remove e destrói todos os elemento da estrutura.

- int MoveFirst ()

  Aponta o elemento corrente para o primeiro elemento da estrutura.

- int MoveNext ()

  Aponta o elemento corrente para próximo elemento da estrutura.

- int MovePrior ()

  Aponta o elemento corrente para o elemento anterior da estrutura.

- int MoveLast ()

  Aponta o elemento corrente para o último elemento da estrutura.

- int GoTo (int pos)

  Aponta o elemento corrente para o elemento na posição `pos`.

- void ResetCurrent ()

  Reseta o elemento corrente da estrutura.

- int Search (T *procurado)

  Busca um elemento na estrutura.

4.20.2 Métodos

4.20.2.1 template<class T> int TSortedList< T >::Count () [inline]

Retorna o numero de elementos da estrutura.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 32 do arquivo TSortedList.h.

```
32 { return TAbstractList<T>::Count(); };
```

4.20.2.2 template<class T> T* TSortedList< T >::Get (int *pos*) [inline]

Retorna um elemento de uma posição específica.

Retorna um ponteiro para elemento da estrutura que ocupa a posição `pos`

Parâmetros:

pos Posição do elemento que se deseja obter

Retorna:

Ponteiro associado ao nó que ocupa a posição `pos` na estrutura

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 39 do arquivo TSortedList.h.

Referenciado por TSortedList< T >::Get().

```
39 { return TAbstractList<T>::Get(pos); };
```

4.20.2.3 template<class T> T* TSortedList< T >::GetCurrent () [inline]

Retorna o elemento corrente da estrutura.

Retorna:

Ponteiro associado ao nó que ocupa a posição corrente na estrutura.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 46 do arquivo TSortedList.h.

Referenciado por TSortedList< T >::Insert().

```
46 { return TAbstractList<T>::GetCurrent(); };
```

4.20.2.4 template<class T> int TSortedList< T >::Insert (T * *data*)

Insere um elemento ordenadamente.

Insere um elemento que aponta para `data` de maneira ordenada na lista

Parâmetros:

data Ponteiro para um elemento que será inserido na estrutura

Retorna:

Posição onde o elemento foi inserido

Exceções:

TENullElement*∗* caso `data` seja nulo (não aponta para nada). **Veja:** TENull-Element

Pós-Condição:

```
data == Get(pos);
```

Complexidade: $O(n)$

Definição na linha 153 do arquivo TSortedList.h.

```
154 {
155         //Posicao onde o elemento deverá estar
156         register int posicao = 0;
157 
158         //Apenas procura a posição se houver varios elementos na lista
159         if(TAbstractList<T>::Count() != 0)
160         {
161                 //Varre cada elemento
162                 MoveFirst();
163 
164                 while(posicao < TAbstractList<T>::Count() )
165                 {
166                         if( *(GetCurrent()) > *data )
167                                 break;
168 
169                         //Avanca
170                         MoveNext();
171                         posicao++;
172                 }
173         }
174 
175         //Retorna a posição
176         return TAbstractList<T>::Insert(data, posicao);
177 }
```

4.20.2.5 template<class T> int TSortedList< T >::Del (int *pos*) [inline]

Remove um elemento da estrutura.

Remove o elemento da estrutura na posição `pos`. Esse método não destrói o elemento apontado pelo nó, apenas retira-o da estrutura, para destruí-lo use Kill()

Parâmetros:

pos Posição do elemento a ser removido da lista.

Retorna:

Posição do elemento corrente da estrutura após a remoção.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 74 do arquivo TSortedList.h.

Referenciado por TSortedList< T >::Del().

```
74 { return TAbstractList<T>::Del(pos); };
```

4.20.2.6 template<class T> void TSortedList< T >::DelAll () [inline]

Remove todos os elemento da estrutura.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 81 do arquivo TSortedList.h.

```
81 { TAbstractList<T>::DelAll(); };
```

4.20.2.7 template<class T> int TSortedList< T >::Kill (int *pos*) [inline]

Remove e destrói um elemento da estrutura.

Libera a memória ocupada pelo elemento na posição `pos` e retira-o da estrutura. Após a chamada desse método o elemento, qualquer referência ao elemento destruído será inválida.

Parâmetros:

pos Posição do elemento a ser destruído.

Retorna:

Posição do elemento corrente da estrutura.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 88 do arquivo TSortedList.h.

Referenciado por TSortedList< T >::Kill().

```
88 { return TAbstractList<T>::Kill(pos); };
```

4.20.2.8 template<class T> void TSortedList< T >::KillAll () [inline]

Remove e destrói todos os elemento da estrutura.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 95 do arquivo TSortedList.h.

```
95 { TAbstractList<T>::KillAll(); };
```

4.20.2.9 template<class T> int TSortedList< T >::MoveFirst () [inline]

Aponta o elemento corrente para o primeiro elemento da estrutura.

Retorna:
Posição do elemento corrente da estrutura (usualmente retorna 0)

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 102 do arquivo TSortedList.h.

Referenciado por TSortedList< T >::Insert().

```
102 { return TAbstractList<T>::MoveFirst(); };
```

4.20.2.10 template<class T> int TSortedList< T >::MoveNext () [inline]

Aponta o elemento corrente para próximo elemento da estrutura.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 109 do arquivo TSortedList.h.

Referenciado por TSortedList< T >::Insert().

```
109 { return TAbstractList<T>::MoveNext(); };
```

4.20.2.11 template<class T> int TSortedList< T >::MovePrior () [inline]

Aponta o elemento corrente para o elemento anterior da estrutura.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 116 do arquivo TSortedList.h.

```
116 { return TAbstractList<T>::MovePrior(); };
```

4.20.2.12 template<class T> int TSortedList< T >::MoveLast () [inline]

Aponta o elemento corrente para o último elemento da estrutura.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 123 do arquivo TSortedList.h.

```
123 { return TAbstractList<T>::MoveLast(); };
```

4.20.2.13 template<class T> int TSortedList< T >::GoTo (int *pos*) [inline]

Aponta o elemento corrente para o elemento na posição `pos`.

Parâmetros:
pos Posição do novo elemento corrente

Retorna:
Posição do novo elemento corrente

Pós-Condição:
PosCorrente = `pos`

Exceções:
TEOutOfBounds caso `pos` seja uma posição inválida

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 130 do arquivo TSortedList.h.

Referenciado por TSortedList< T >::GoTo().

```
130 { return TAbstractList<T>::GoTo(pos); };
```

4.20.2.14 template<class T> void TSortedList< T >::ResetCurrent () [inline]

Reseta o elemento corrente da estrutura.

Complexidade: $O(1)$

Reimplementação de TAbstractList< T >.

Definição na linha 137 do arquivo TSortedList.h.

```
137 { TAbstractList<T>::ResetCurrent(); };
```

4.20.2.15 template<class T> int TSortedList< T >::Search (T * *procurado*) [inline]

Busca um elemento na estrutura.

Busca o primeiro elemento da estrutura que seja igual a `procurado` e retorna sua posição.

Parâmetros:
procurado Elemento a ser buscado na estrutura

Retorna:
Posição do elemento, caso tenha encontrado algum, ou -1 caso o elemento não tenha sido encontrado.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 144 do arquivo TSortedList.h.

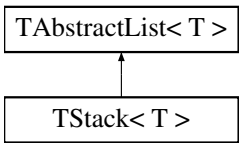
Referenciado por TSortedList< T >::Search().

```
144 { return TAbstractList<T>::Search(procurado); };
```

4.21 Referência da Template de Classe TStack< T >

`#include <TStack.h>`

Diagrama de Hierarquia para TStack< T >::



4.21.1 Descrição Detalhada

template<class T> class TStack< T >

Lista genérica.

Implementa listas genéricas.

Definição na linha 20 do arquivo TStack.h.

Métodos Públicos

- int Count ()
  Retorna o numero de elementos da pilha.
- T * Top ()
  Retorna o dado do elemento do topo da pilha.
- T * Base ()
  Retorna o dado do elemento da base da pilha.
- T * Pop ()
  Desempilha um elemento.

- int Push (T *data)
  Empilha um elemento.
- T * Get (int pos)
  Retorna um elemento de uma posição específica.
- int Search (T *procurado)
  Busca um elemento na estrutura.
- void DelAll ()
  Remove todos os elemento da pilha.
- void KillAll ()
  Destrói todos os elemento da pilha.

4.21.2 Métodos

4.21.2.1 template<class T> int TStack< T >::Count () [inline]

Retorna o numero de elementos da pilha.

Veja também:
TAbstractList::Size

Reimplementação de TAbstractList< T >.

Definição na linha 31 do arquivo TStack.h.

```
31 { return TAbstractList<T>::Count(); };
```

4.21.2.2 template<class T> T* TStack< T >::Top () [inline]

Retorna o dado do elemento do topo da pilha.

Retorna:
Dado do elemento do topo da pilha

Definição na linha 43 do arquivo TStack.h.

Referenciado por TStack< T >::Pop().

```
43 { TAbstractList<T>::MoveFirst(); return TAbstractList<T>::GetCurrent(); };
```

4.21.2.3 template<class T> T* TStack< T >::Base () [inline]

Retorna o dado do elemento da base da pilha.

Retorna:
Dado do elemento da base da pilha

Definição na linha 55 do arquivo TStack.h.

```
55 { TAbstractList<T>::MoveLast(); return TAbstractList<T>::GetCurrent(); };
```

4.21.2.4 template<class T> T* TStack< T >::Pop () [inline]

Desempilha um elemento.

Remove o elemento do topo da pilha e retorna-o

Retorna:
Dado do elemento do topo da pilha

Exceções:
TEStackUnderflow caso a pilha esteja vazia

Veja também:
TEGeneralError

Definição na linha 72 do arquivo TStack.h.

```
72 { if(TAbstractList<T>::Count() == 0) { throw new TEStackUnderflow("TStack::Pop"); } T* retorno =
```

4.21.2.5 template<class T> int TStack< T >::Push (T * *data*) [inline]

Empilha um elemento.

Empilha um elemento no topo da pilha e retorna a posição do topo da pilha

Parâmetros:
data Ponteiro para um elemento que será inserido na lista

Retorna:
Posição do topo da Pilha

Exceções:
TENullElement caso `data` seja nulo (não aponta para nada). **Veja:** TENullElement

Veja também:
TEGeneralError

Definição na linha 90 do arquivo TStack.h.

Referenciado por TStack< T >::Push().

```
90 { return TAbstractList<T>::Insert(data, 0); };
```

4.21.2.6 template<class T> T* TStack< T >::Get (int *pos*) [inline]

Retorna um elemento de uma posição específica.

Retorna um ponteiro para elemento da estrutura que ocupa a posição `pos`

Parâmetros:

pos Posição do elemento que se deseja obter

Retorna:

Ponteiro associado ao nó que ocupa a posição `pos` na estrutura

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 97 do arquivo TStack.h.

Referenciado por TStack< T >::Get().

```
97 {return TAbstractList<T>::Get(pos); };
```

4.21.2.7 template<class T> int TStack< T >::Search (T * *procurado*) [inline]

Busca um elemento na estrutura.

Busca o primeiro elemento da estrutura que seja igual a `procurado` e retorna sua posição.

Parâmetros:

procurado Elemento a ser buscado na estrutura

Retorna:

Posição do elemento, caso tenha encontrado algum, ou -1 caso o elemento não tenha sido encontrado.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 104 do arquivo TStack.h.

Referenciado por TStack< T >::Search().

```
104 { return TAbstractList<T>::Search(procurado); } ;
```

4.21.2.8 template<class T> void TStack< T >::DelAll () [inline]

Remove todos os elemento da pilha.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 116 do arquivo TStack.h.

```
116 { TAbstractList<T>::DelAll(); };
```

4.21.2.9 template<class T> void TStack< T >::KillAll () [inline]

Destrói todos os elemento da pilha.

Complexidade: $O(n)$

Reimplementação de TAbstractList< T >.

Definição na linha 128 do arquivo TStack.h.

```
128 { TAbstractList<T>::KillAll(); };
```

Index