Estruturas de Dados

Trabalho 1 – Listas, Filas, Pilhas e Hash

Fabrício V. Matos

Versão draft 0

1. Errata

-

2. Regras do Jogo

- As regras aqui definidas têm precedência sobre eventuais regras apresentadas informalmente em sala de aula;
- O trabalho deve ser desenvolvido individualmente ou em duplas;
- Data/Hora limite de entrega: 30/05/2005 07:00:00 (segunda-feira às 7 da manhã)
- Estarão sujeitos a não serem corrigidos, recebendo nota zero, trabalhos:
  - Entregues fora do prazo;
  - Incompletos (faltando arquivos ou classes);
  - Com erro de compilação;
  - Com erro de execução grave (aborta ou não exibe resultado);
- Cada função deve ter uma indicação (comentário) de sua complexidade de tempo: O(n), O(1), etc.;
- Será avaliado:
  - 50% – Correção (todos os procedimentos se compartam da forma esperada);
  - 20% – Apresentação (identação, nomes de variáveis, etc.) e clareza do código (códigos bem legíveis e comentados);
  - 10% – Criatividade. Surpreendam-me!
  - 20% – Eficiência (performance)
- É importante destacar em sua documentação (arquivo `ed20051-trab1.pdf` ou `ed20051-trab1.txt`) questões relevantes relacionadas à elegância de suas soluções para evitar que ela possa vir a passar desapercebida;
- Formato de entrega:
  - Um único arquivo "`ed20051-trab1.tar.gz`" anexado a um email:
    - Destino: `fabricio@qualidata.com.br`
    - Assunto: `ed20051-trab1`
    - Conteúdo: Na primeira linha os nomes com "." no lugar do espaço e separados por "&" (caso o trabalho seja em grupo). Na segunda linha os emails separados por &. Exemplo:

      ```
      Trifosa.da.Silva&Trifena.Pereira
      trifosa@uol.com.br&trifenapereira@hotmail.com
      ```
  - O arquivo deve conter apenas os arquivos: *.cpp, *.h, makefile e, opcionalmente, *.pdf (ou *.txt) contendo a documentação do trabalho.
  - As regras a serem implementadas no arquivo `makefile` serão definidas mais adiante.
  - Após enviar o email, você deverá receber uma mensagem (não automática) confirmando o recebimento do email. Só após receber tal confirmação você deve assumir que o trabalho foi entregue.
- Para todos efeitos, a data/hora efetiva de entrega do trabalho será a data fornecida pelo servidor mail.qualidata.com.br que constará no cabeçalho do e-mail.

3. Hierarquia de Classes

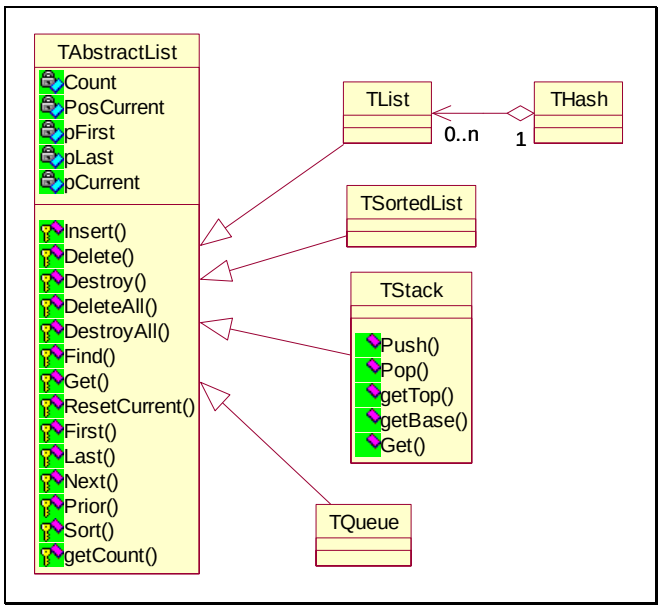

Figura 1 – Diagrama de Classes

É obrigatória a implementação de todas essas classes, porém seus métodos e atributos podem ser alterados, desde que o comportamento esperado para cada classe seja mantido. A seguir apresentaremos a descrição de cada classe. Os métodos/atributos eventualmente apresentados são apenas sugestões. Além dessas classes devem existir ainda classes de exceções (pelo menos uma) que deverão ser utilizadas para abortar a execução quando alguma restrição de integridade for violada.

4. Makefile e Programa Principal

Deverá haver uma regra `all` no Makefile que ao ser executada (`[]# make all`) compile todo o programa e gere um arquivo executável de nome `ei20042-trab1`.

Além dos arquivos `*.h/*.cpp` referentes a cada classe, também deverá ser fornecido um arquivo `main.cpp` que conterá a implementação da função `main()` (função principal do programa C++) que deverá ter precisamente o seguinte comportamento:

a. **`[]# ./ei20042-trab1 –l input.txt searches.txt`**

- Parâmetro "-l" indica que deve ser utilizada a TList.
- input.txt : nome do arquivo que conterá os dados (código e nome) das Pessoas a serem inseridas na lista. Formato[1]:

Definição	Exemplo (input.txt)

`<posição>TAB<código>TAB<nome>`
`<posição>TAB<código>TAB<nome>`
`...`
	`0 1342 Fulano de Tal`
	`1 442 Ciclano de Tal`
	`...`

- searches.txt: nome de um segundo arquivo que conterá as buscas (por código) que devem ser executadas. Este arquivo deve ter o seguinte formato:

[1] *Em cada linha deve haver uma pessoa seguindo o seguinte formato:* `printf("%d\t%d\t%s\n", pos, codigo, nome)`

Definição	Exemplo (searches.txt)
<Número de Buscas N> <1º Código a ser buscado> <2º Código a ser buscado> ... <N-ésimo Código a ser buscado>	4 1342 2654 9856 3442

- Saída: O sistema deverá imprimir no stdout (via `printf`, p.e.), na mesma ordem da entrada, os códigos fornecidos para busca, seguidos do nome da pessoa encontrada. Casa não encontre alguma pessoa com o código indicado, deve retornar *"[Item not found]"* no lugar de seu nome. Ao final deve imprimir a quantidade de elementos não encontrados conforme o exemplo abaixo:

Entrada (searches.txt)	Saída (stdout)
4 1342 2654 9856 3442	1342 – Fulano de Tal 2654 – [Item not found] 9856 – [Item not found] 3442 – Ciclano de Tal e Tal => 2 item(s) not found.

b. **`[]# ./ei20042-trab1 –sl input.txt searches.txt`**
O mesmo que o parâmetro "`-l`", mas utilizando `TSortedList`. Logo a posição indicada no arquivo `input.txt` embora continue existindo, deve ser ignorada.

c. **`[]# ./ei20042-trab1 –s input.txt searches.txt`**
O mesmo que o parâmetro "`-l`", mas utilizando `TStack`. Logo a posição indicada no arquivo `input.txt` deve ser ignorada.

d. **`[]# ./ei20042-trab1 –q input.txt searches.txt`**
O mesmo que o parâmetro "`-l`", mas utilizando `TQueue`. Logo a posição indicada no arquivo `input.txt` deve ser ignorada.

e. **`[]# ./ei20042-trab1 –h input.txt searches.txt`**
O mesmo que o parâmetro "`-l`", mas utilizando `THash`. Logo a posição indicada no arquivo `input.txt` deve ser ignorada.

5. TAbstractList

Classe que implementa uma lista genérica duplamente encadeada que servirá de base (ancestral) para a implementação de Listas, Filas e Pilhas genéricas.

```
struct node {
  TYPE *Data;
  item *pNext,*pPrior;
} TNode
```

a. Classe de Exceção (sugestão)

```
class TAbstractListError: public std::exception {
  public:
    TAbstractListError (char *msg) { printf("[AbstractListError] %s\n",
msg);
    };
};
```

b. Membros Públicos da Classe (sugestão)

Nenhum.

c. Membros Protegidos da Classe (sugestão)

TAbstractList();
- *Método construtor.*

~TAbstractList();
- *Método destrutor.*
- *Deve liberar a lista - equivale a executar DeleteAll()*

int Insert(int pos, TYPE *data);
- *Insere o elemento "data" na posição "pos" e incrementa FCount*
- *Após a inserção, deve ser verdade: data == This()->Get(pos)*
- *Se bem sucedido, deve retorna "pos"*
- *Insert em uma posição inválida (pos<0 ou pos >FCount) gerar exceção*
- *Atualizar o elemento corrente para o elemento inserido*
- *Se data == NULL, gerar exceção*

int Delete(int pos);
- *Delete em uma posição inválida (pos<0 ou pos >=FCount) gerar exceção*
- *Decrementa FCount*
- *Ao deletar um elemento (delete pItem), FPosCurrent não deve mudar, ou seja, o elemento seguinte passa a ser o corrente. Exceção: Se for o ultimo, o anterior passa a ser o corrente. Exceção: Se houver apenas 1 elemento, fazer pCurrent = pFirst = pLast = NULL e FposCurrente = -1*
- *Deve retornar FposCurrent*

int Destroy(int pos);
- *Libera o dado apontado pelo elemento em questão - free(pItem->Data)*
- *Deleta o elemento da lista (equivale a This()->Delete(pos)), seguindo os mesmos critérios de Delete()*
- *O retorno segue os mesmos critérios de Delete()*

void DeleteAll();
- *Exclui todos os elementos da lista. Equivale a chamar Delete() para todos os elementos da lista.*

void DestroyAll();
- *Destroy todos os elementos da lista. Equivale a chamar Destroy() para todos os elementos da lista.*

int Find(TYPE *data);
- *Se data for NULL, levantar uma exceção;*
- *Busca o elemento na lista;*
- *Ao encontrar a primeira ocorrência, atualiza o elemento corrente e retorna a posição do elemento.*
- *Se não encontrar, reseta o elemento corrente (corrente = NULL) e retorna -1*

void ResetCurrent();
- *Elemento corrente passa a ser NULL (pCurrent = NULL e FPosCurrent = -1)*

TYPE *Get(int pos);
- *Faz o elemento corrente ser o da posição "pos"*
- *Retorna pCurrent->Data*
- *Se posição for inválida (pos<0 ou pos>=FCount), deve abortar (levantar uma exceção – throw ...)*

TYPE *GetCurrent();
- *Retorna pCurrent->Data;*
- *se pCurrent=NULL, levantar uma exceção;*

int First();
- *Atualiza o elemento corrente para a primeira posição e retorna FPosCurrent*
- *Se a lista estiver vazia, retorna -1*

int Last();
- *Atualiza o elemento corrente para a última posição e retorna FPosCurrent*
- *Se a lista estiver vazia, retorna -1*

int Next();
- *Atualiza pCurrent para pCurrent->pNext e faz FPosCurrent++*
- *Se já estiver na última posição, reseta - equivale a ResetCurrent() e retorna -1*
- *Se estiver resetado (pCurrent = NULL), equivale a First()*

int Prior();
- *Atualiza pCurrent para pCurrent->pPrior e faz FPosCurrent--*
- *Se já estiver na primeira posição, reseta - equivale a ResetCurrent() e retorna -1*
- *Se estiver resetado (pCurrent = NULL), equivale a Last()*

int getCount();
- *Retorna FCount*

void Sort();
- *Ordena a lista;*
- *Reseta o elemento corrente: equivale a ResetCurrent()*

d. Membros Privados da Classe (sugestão)

```
TNode *pFirst, *pLast, *pCurrent; //Ponteiros para os itens da lista
int FCount, FPosCurrent; //Variáveis de controle
```

6. TList

Classe que simplesmente torna públicos os métodos protegidos de TAbstractList e Implementa listas genéricas.

7. TStack

Classe que implementa pilha genérica herdando e utilizando métodos protegidos de TAbstractList. A seguir apresentamos sugestões de membros (métodos e atributos) da classe TStack.

a. Membros Públicos da Classe (sugestão)

`TStack();`
- *Deve chamar o construtor de TAbstractLinkedList passando NULL como função de comparação*

`~TStack();`
- *Chama (implicitamente) o destrutor de TAbstractLinkedList*

`int Push(TYPE *data);`
- *Empilha e retorna a posição do topo*
- *A pilha não tem limite pré-definido (limitada apenas pelos recursos do próprio computador)*

`TYPE *Pop();`
- *Se a pilha estiver vazia, deve gerar uma exceção:*
  `throw TStackError("Stack Underflow");`
- *Desempilha o elemento do topo da pilha*
- *Retorna o elemento que foi desempilhado*

`TYPE *getTop();`
- *Retorna o dado do elemento do topo da pilha*
- *Se a pilha estiver vazia, deve retornar NULL*

`TYPE *getBase();`
- *Retorna o dado do elemento da base da pilha*
- *Se a pilha estiver vazia, deve retornar NULL*

`void DeleteAll();`
- *Publica o método equivalente em TAbstractLinkedList*

`void DestroyAll();`
- *Publica o método equivalente em TAbstractLinkedList*

`int getCount();`
- *Publica o método equivalente em TAbstractLinkedList*

b. Membros Protegidos da Classe (sugestão)
Nenhum.

c. Membros Privados da Classe (sugestão)
Nenhum.

8. TQueue

Classe que implementa filas (FIFO) genéricas herdando e utilizando métodos protegidos de TAbstractList. Seguem as sugestões de membros da classe TQueue.

a. Membros Públicos da Classe (sugestão)

`TQueue();`
- *Deve chamar o construtor de TAbstractList*

`~TQueue();`
- *Chama (implicitamente) o destrutor de TAbstractList*

`int Append (TYPE *data);`
- *Insere o elemento no final da fila: equivale a Insert(getCount(), data)*

`int Delete();`

- *Exclui o primeiro da fila: Equivale a TAbstractList::Delete(0)*

int Destroy();
- *Destrói o primeiro da fila: Equivale a TAbstractList::Destroy(0)*

void DeleteAll();
- *Publica o método equivalente em TAbstractList*

void DestroyAll();
- *Publica o método equivalente em TAbstractList*

int Find(TYPE *data);
- *Publica o método equivalente em TAbstractList*

void ResetCurrent();
- *Publica o método equivalente em TAbstractList*

TYPE *Get(int pos);
- *Publica o método equivalente em TAbstractList*

TYPE *GetCurrent();
- *Publica o método equivalente em TAbstractList*

int First();
- *Publica o método equivalente em TAbstractList*

int Last();
- *Publica o método equivalente em TAbstractList*

int Next();
- *Publica o método equivalente em TAbstractList*

int Prior();
- *Publica o método equivalente em TAbstractList*

int Count();
- *Publica o método equivalente em TAbstractList*

b. Membros Protegidos da Classe (sugestão)
Nenhum.

c. Membros Privados da Classe (sugestão)
Nenhum.

9. TSortedList

Listas que sempre inserem os novos elementos em sua posição ordenada. Assim, as funções de inserção dessa classe não devem pedir a posição de inserção. Assim, todos os métodos de TAbstractList devem ser publicados, exceto "Sort" que não será necessário e o "Insert" que será alterado para:

int Insert(TYPE *data);
- *Insere o elemento "data" na posição ordenada*
- *Se bem sucedido, deve retorna "pos" (sua posição na lista)*
- *Atualizar o elemento corrente para o elemento inserido*
- *Se data == NULL, gerar exceção*

10. THash

Classe que implementa uma tabela hash genérica. Ela não descende de TAbstractList. Internamente será utilizada a classe TList para guardar os elementos inseridos na mesma posição da tabela (quando houver colisão de chaves).

a. Classe de Exceção (sugestão)

```
class THashError: public std::exception {
  public:
    THashError (char *msg) { printf("[HashError] %s\n", msg);
    };
};
```

b. Membros Públicos da Classe (sugestão)

THash(int (*fnHash)(void*, int), int Size);

- *Deve criar um vetor de "Size" posições com um Tlist em cada uma delas;*
- *Se fnHash for NULL, deve gerar uma exceção:*
  throw THashError ("*fnHash* is required");
- *Se Size < 1, deve gerar uma exceção:*
  throw THashError ("*Size must be positive (Size > 0)*");

~THash();

- *Deleta todas as listas criadas*
- *Desaloca toda memória alocada (tabela)*

int Add(TYPE *data);

- *Insere o elemento na tabela hash:*
  - *Calcula a chave hash: k = pFnHash(data, FSize)*
  - *Adiciona o elemento à lista: V[k]->Insert(0, data)*
- *Retorna a posição na tabela hash em que foi inserido o elemento – equivale a retornar pFnHash(data)*
- *Se em algum momento a função fnHash retornar um valor inválido (valor < 0 ou valor >= Size) deve ser gerada uma exceção:*
  throw THashError ("Invalid hash function");
- *Se data == NULL* => throw THashError ("Can´t add NULL element");

bool Delete(TYPE *data);

- *Exclui da tabela o elemento cuja chave de busca seja igual à chave em "data":*
  - *Calcula a chave hash: k = pFnHash(data, FSize)*
  - *Localiza o elemento na lista: i = V[k]->Find(data)*
  - *Exclui o elemento da lista: V[k]->Delete(i)*
- *Se tal elemento existir na lista, retorna true, senão, retorna false.*
- *Se data == NULL* => throw THashError ("data excpected in Delete(data)");

int Destroy(TYPE *data);

- *Semelhante à Delete()*
- *Libera memória do dado do elemento excluído: delete elemento->Data*
- *Se data == NULL* => throw THashError ("data excpected in Destroy(data)");

void DeleteAll();

- *Chama DeleteAll() das listas de cada posição da tabela hash.*

void DestroyAll();

- *Chama DestroyAll() das listas de cada posição da tabela hash.*

`TYPE *Get(TYPE *data);`

- *Localiza na tabela o elemento cuja chave de busca seja igual à chave em "data", retornando tal elemento.*
- *Se não for encontrado, retorna NULL*
- *Se data == NULL =>* `throw THashError ("data excpected in Get(data)");`

`int getCount();`

- *Retorna a quantidade total de elementos na tabela hash*

`int getSize();`

- *Retorna FSize*

c. Membros Protegidos da Classe (sugestão)

Nenhum.

d. Membros Privados da Classe (sugestão)

```
int (*pFnHash)(void*, int); //Ponteiro para a função hash
int FSize, FCount;          //Campos (fields) p/ guardar dados
TList **V;                  //Tabela (não alocada) de listas
```

Nota 1: Esta é uma versão rascunho, sujeita a críticas e modificações;

Nota 2: Qualquer eventual erro ou discrepância encontrada nesta especificação deve ser notificado ao professor imediatamente para que seja corrigida;