**Projecto da Disciplina de Laboratórios de Informática III**

Grupo Nº 73

|  |  |  |
| --- | --- | --- |
|  | João Carlos Faria Padrão | Aluno nº: 76438 |
|  | Rafael Dias Alves | Aluno nº: 72629 |
|  | Ricardo Jorge Silva Braga Marques | Aluno nº: 55723 |

**Indice**

**Introdução**

No âmbito da Disciplina de laboratórios de informática III, foi-nos pedido o desenvolvimento de um aplicativo em Linguagem C que trabalhasse com um grande volume de dados.

Foi-nos pedido que tal aplicativo tivesse modularidade e encapsulamento.

De seguida serão apresentados os vários módulos que caracterizam a nossa aplicação.

**Modulo main**

No modulo main encontra-se o menu de seleção das queries a serem executadas.

Os dados são carregados automaticamente quando o programa arranca, havendo a possibilidade de os recarregar.

Depois de escolhida a opção pretendida, a query é executada, sendo apresentado no final da execução o tempo que demorou em segundos

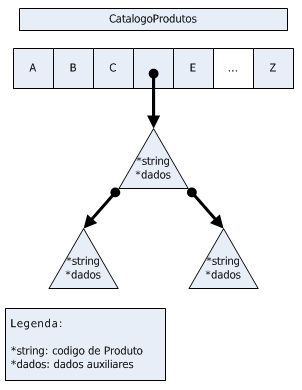
**Módulo Produtos**

O módulo de Produtos caracteriza-se por uma API com tipos de dados encapsulados onde constam as funções responsáveis por criar e actualizar as listas de produtos da aplicação.

A estrutura de dados principal divide-se em duas partes distintas, distribuídas pelos ficheiros API-Produtos.c e API-Produtos.h e assentam numa AVL balanceada, por questões de eficiência. (ver AVL).

No ficheiro API-Produtos.c temos a seguinte estrutura de dados:

|  |
| --- |
| #define TAMCAT 26  #define MINP 1000  #define MAXP 1999  typedef struct StructProdutos{  BTree lista[TAMCAT];  int tamanho[TAMCAT];  int crescimento[TAMCAT];  } \*listaprodutos; |



A estrutura StructProdutos é composta por 26 AVL’s, onde cada AVL é responsável por armazenar os Produtos cujo código se inicia por uma letra de A a Z respectivamente. A variável tamanho indica o numero de elementos que estão contidos em cada uma das AVL’s. A variável crescimento é responsável pelo balanceamento da AVL.

No ficheiro API-Produtos.h temos as seguintes estruturas de dados:

|  |
| --- |
| typedef char\* Produto;  typedef char\*\* ListaProdutos;  typedef struct StructProdutos \*CatalogoProdutos; |

Produto é um código de produto composto por uma string.

ListaProdutos é uma listagem de todos os produtos existentes por ordem alfabética entre AA1000 e ZZ1999.

CatalogoProdutos é uma estrutura baseada em AVL que contem os produtos válidos existentes no sistema. No fundo é uma forma de podermos aceder à struct StructProdutos sem que o utilizador de aperceba de como os dados estão guardados, garantindo assim o encapsulamento do tipo de dados

Da API de produtos fazem parte as seguintes funções:

* CatalogoProdutos iniciaCatProdutos();
  + Função responsável por iniciar um catalogo de Produtos.
  + Retorna um tipo CatalogoProdutos.
* CatalogoProdutos copiaCatProdutos(CatalogoProdutos catProd);
  + Função responsável por copiar um catalogo de Produtos.
  + Recebe um tipo CatalogoProdutos.
  + Retorna um tipo CatalogoProdutos que é uma cópia do CatalogoProdutos recebido.
* CatalogoProdutos insereProduto(CatalogoProdutos catProd, Produto p);
  + Função responsável por inserir um novo código de produtos num Catalogo de Produtos.
  + Recebe um tipo CatalogoProdutos.
  + Recebe um tipo Produto que é um código de produto.
  + Retorna um tipo CatalogoProdutos.
* Boolean existeProduto(CatalogoProdutos catProd, Produto p);
  + Função que verifica se um produto existe dentro de um catalogo.
  + Recebe um tipo CatalogoProdutos.
  + Recebe um Produto que é um código de produto.
  + Retorna um booleano, TRUE ou FALSE.
* int totalProdutos(CatalogoProdutos catProd);
  + Função que calcula o total de produtos de um Catalogo.
  + Recebe um tipo CatalogoProdutos.
  + Retorna um inteiro que é o total de produtos contidos no catalogo.
* int totalProdutosLetra(CatalogoProdutos catProd, char letra);
  + Função que calcula o total de produtos de um catalogo, iniciado por determinada Letra.
  + Recebe um tipo CatalogoProdutos.
  + Recebe um Caracter.
  + Retorna um inteiro que é o total de produtos iniciado pelo caracter recebido.
* void removeCatProdutos(CatalogoProdutos catProd);
  + Função que elimina um catalogo de produtos e faz free() ao tipo CatalogoProdutos
  + Recebe um tipo CatalogoProdutos.
* void removeCatProdutos2(CatalogoProdutos catProd);
  + Função que elemina um catalogo de produtos e não faz free() ao tipo CatalogoProdutos.
  + Recebe um tipo CatalogoProdutos.
* void\* retornaDadosProduto(CatalogoProdutos catProd, Produto p);
  + Função que retorna um apontador para os dados de um produtos. (ver AVL)
  + Recebe um tipo CatalogoProdutos.
  + Recebe um Produto que é um código de produto.
  + Retorna um apontador para os dados detalhados de um produto dado.
* void insereDadosProduto(CatalogoProdutos catProd, Produto p, void \*dados);
  + Função que actualiza o apontador para os dados detalhados de um produtos.
  + Recebe um tipo CatalogoProdutos.
  + Recebe um Produto que é um código de produto.
  + Recebe um apontador para um tipo de dados genérico. (ver AVL)
* char\*\* imprimeProdutosLetra (CatalogoProdutos catProd , char letra, int\* j);
  + Função que retorna um array com todos os codigos de produto começados por uma letra.
  + Recebe o catalogo de produtos e a letra pretendida.
  + Retorna um array de strings com os codigos de Produto.
* void retornaProdutos(CatalogoProdutos catProd , char\*\* str, int\* j);
  + Função
* ListaProdutos retornaListaProdutos (CatalogoProdutos catProd);
  + Função que retorna uma listagem de todos os produtos contidos no catálogo de Produtos.
  + Recebe um tipo Catalogo de Produtos.
  + Retorna um tipo Lista de Produtos.
* void removeListaProdutos(ListaProdutos ListP, int tamanho);
  + Função que elemina uma listagem de todos os produtos contidos no catálogo de Produtos.
  + Recebe um tipo Lista de Produtos.
  + Recebe um inteiro que é o tamanho da Listagem.

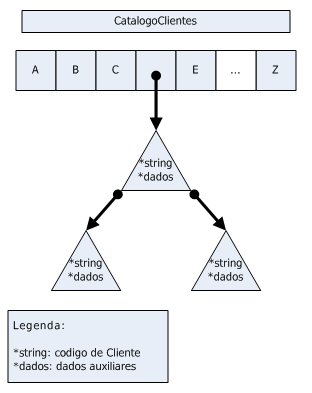
**Módulo Clientes**

O módulo de Clientes caracteriza-se por uma API com tipos de dados encapsulados onde constam as funções responsáveis por criar e actualizar as listas de clientes da aplicação.

A estrutura de dados principal divide-se em duas partes distintas, distribuídas pelos ficheiros .API-Clientes.C e API-Clientes.h e assentam numa AVL balanceada, por questões de eficiência. (ver AVL).

No ficheiro API-Clientes.C temos a seguinte estrutura de dados:

|  |
| --- |
| #define TAMCAT 26  #define MINCLIENTE 1000  #define MAXCLIENTE 5000  typedef struct StructClientes{  BTree lista[TAMCAT];  int tamanho[TAMCAT];  int crescimento[TAMCAT];  } \*structclientes; |



A estrutura StructClientes é composta por 26 AVL’s, onde cada AVL é responsável por armazenar os clientes cujo código se inicia por uma letra de A a Z respectivamente. A variável tamanho indica o numero de elementos que estão contidos em cada uma das AVL’s. A variável crescimento é responsável pelo balanceamento da AVL.

No ficheiro API-Clientes.h temos as seguintes estruturas de dados:

|  |
| --- |
| typedef char\* Cliente;  typedef char\*\* ListaClientes;  typedef struct StructClientes \*CatalogoClientes; |

Cliente é um código de cliente composto por uma string.

ListaClientes é uma listagem de todos os clientes existentes por ordem alfabética entre A1000 e Z5000.

CatalogoClientes é uma estrutura baseada em AVL que contem os clientes válidos existentes no sistema. No fundo é uma forma de podermos aceder à struct StructClientes sem que o utilizador de aperceba de como os dados estão guardados, garantindo assim o excapsulamento do tipo de dados

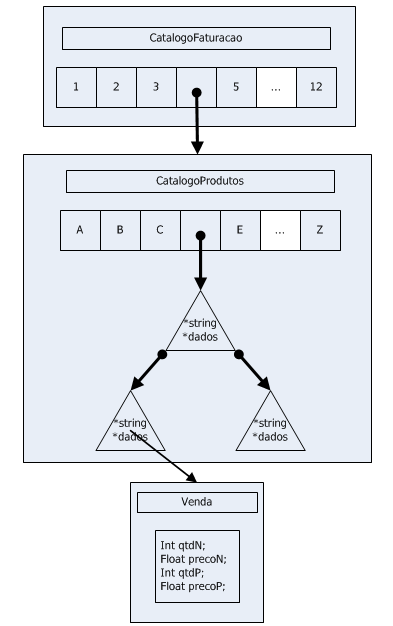
Da API de clientes fazem parte as seguintes funções:

* CatalogoClientes iniciaCatClientes();
  + Função responsável por iniciar um novo catalogo de clientes.
  + Retorna um tipo CatalogoClientes.
* CatalogoClientes copiaCatClientes(CatalogoClientes catCli);
  + Função responsável por criar uma copia de um catalogo de clientes existente.
  + Recebe um tipo CatalogoClientes.
  + Retorna um tipo CatalogoClientes que é uma copia do catalogo recebido.
* CatalogoClientes insereCliente(CatalogoClientes catCli, Cliente c);
  + Função que insere um novo código de cliente num catalogo.
  + Recebe um tipo CatalogoClientes.
  + Recebe um tipo Cliente que é um código de cliente.
  + Retorna um tipo CatalogoClientes.
* Boolean existeCliente(CatalogoClientes catCli, Cliente c);
  + Função que verifica se determinado cliente existe num catalogo.
  + Recebe um tipo CatalogoClientes.
  + Recebe um tipo Cliente que é um código de cliente.
  + Retorna um booleano que pode ser TRUE ou FALSE.
* int totalClientes(CatalogoClientes catCli);
  + Função que calcula o numero total de clientes de um catalogo.
  + Recebe um tipo CatalogoClientes.
  + Retorna um inteiro que é o numero total de clientes no catalogo.
* int totalClientesLetra(CatalogoClientes catCli, char letra);
  + Função que calcula o numero total de clientes iniciado por determinada letra.
  + Recebe um tipo CatalogoClientes
  + Recebe um caracter
  + Retorna um inteiro que é o numero de clientes iniciado pelo caracter recebido.
* void removeCatClientes(CatalogoClientes catCli);
  + Função que elimina um catalogo de clientes e faz free() ao tipo CatalogoClientes.
  + Recebe um tipo CatalogoClientes.
* void removeCatClientes2(CatalogoClientes catCli);
  + Função que elimina um catalogo de clientes e não faz free() ao tipo CatalogoClientes.
  + Recebe um tipo CatalogoClientes.
* void\* retornaDadosCliente(CatalogoClientes catCli, Cliente c);
  + Função que retorna um apontador para os dados detalhados de um cliente (verAVL).
  + Recebe um tipo CatalogoClientes.
  + Recebe um tipo Cliente que é um código de cliente.
  + Retorna um apontador para os dados detalhados de um cliente.
* void insereDadosCliente(CatalogoClientes catCli, Cliente c, void\* dados);
  + Função que insere os dados detalhados de um cliente.(verAVL)
  + Recebe um tipo CatalogoClientes.
  + Recebe um tipo Cliente que é um código de cliente.
  + Recebe um apontador para os dados detalhados de um cliente.(ver AVL)
* ListaClientes retornaListaClientes (CatalogoClientes catCli);
  + Função que retorna uma listagem dos códigos de cliente existentes num catalogo
  + Recebe um tipo CatalogoClientes.
  + Retorna um tipo ListaClientes.
* void removeListaClientes(ListaClientes ListC, int tamanho);
  + Função que remove uma listagem dos códigos de cliente.
  + Recebe um tipo ListaClientes
  + Recebe um inteiro que é o tamanho da listagem de clientes.

**Modulo Facturação**

O módulo de Facturação caracteriza-se por uma API com tipos de dados encapsulados onde constam as funções responsáveis por criar, actualizar e consultar a informação referente ás vendas efectuadas pela empresa.

A estrutura de dados principal divide-se em duas partes distintas, distribuídas pelos ficheiros .API-Faturacao.C e API-Faturacao.h e assentam numa AVL balanceada, por questões de eficiência. (ver AVL).

No ficheiro API-Faturacao.C temos as seguintes estruturas de dados:

|  |
| --- |
| #define FMTAM 12  #define FCTAM 26  #define MINCLIENTE 1000  #define MAXCLIENTE 5000  #define MINPRODUTO 1000  #define MAXPRODUTO 1999  typedef struct ListaFaturacao{  CatalogoProdutos catMes[FMTAM];  int vendasValidas[FMTAM];  } lf1;    typedef struct Venda{  int qtdN;  float precoN;  int qtdP;  float precoP;  } \*venda; |

A estrutura ListaFaturacao é composta por doze Catalogos de produtos, cada um representando um mês do ano, onde serão armazenas as vendas referentes a cada produto. Contem um inteiro vendasValidas que é onde se guardam os totais de todas as vendas consideradas válidas.

A estrutura Venda é um dado auxiliar, carregado dentro de cada produto onde se inserem as quantidades vendidas e os preços, quer em modo normal, quer em modo promoção.(ver modulo Produtos e AVL)

No ficheiro API-Faturacao.h temos a seguinte estrutura de dados:

|  |
| --- |
| typedef struct ListaFaturacao \*CatalogoFaturacao; |

CatalogoFaturacao é uma estrutura baseada em AVL que contem as vendas válidas existentes no sistema. No fundo é uma forma de podermos aceder à struct ListaFaturacao sem que o utilizador de aperceba de como os dados estão guardados, garantindo assim o excapsulamento do tipo de dados.

Da API de Faturacao fazem parte as seguintes funções:

* CatalogoFaturacao iniciaCatFaturacao(CatalogoProdutos catProd, int nfiliais);
  + Função responsável por iniciar um catalogo de facturação baseado num catalogo de produto pre-existente.
  + Recebe um tipo CatalogoProdutos.
  + Recebe um inteiro que é igual ao numero de filiais do sistema.
  + Retorna um tipo CatalogoFaturacao
* CatalogoFaturacao insereCompraFaturacao(CatalogoFaturacao catFact, Produto p, int qtd, float preco, int mes, char tipo, int filial);
  + Função responsável por inserir uma venda validada no catalogo de facturação.
  + Recebe um tipo CatalogoFaturacao.
  + Recebe um tipo Produto que é um código de produto.
  + Recebe um inteiro que é a quantidade vendida.
  + Recebe um float que é o preço de venda.
  + Recebe um inteiro entre 1 e 12 que é o mês em que foi efectuada a venda.
  + Recebe um caracter de entre dois possíveis (N ou P) representando se a venda foi feita em modo normal ou em promoção
  + Recebe um inteiro que é o numero da filial onde foi efectuada a venda.
* void removeCatFaturacao(CatalogoFaturacao catFact, int nfiliais);
  + Função que elimina um catalogo de facturação e faz free() do tipo CatalogoFaturacao.
  + Recebe um tipo CatalogoFaturacao.
  + Recebe um inteiro que é o numero de filiais do sistema.
* int totalVendas(CatalogoFaturacao catFact, int nfiliais);
  + Função que calcula o total de artigos vendidos em todas as filiais.
  + Recebe um tipo CatalogoFaturacao.
  + Recebe um inteiro que é o numero de filiais do sistema.
  + Retorna um inteiro que é o numero total de artigos vendidos.
* int quantidadeVendida(CatalogoFaturacao catFact, int mes , Produto p , int filial , int np);
  + Função que dados um mês, um produto, uma filial e um tipo de venda, calcula a quantidade vendida.
  + Recebe um tipo CatalogoFaturacao
  + Recebe um inteiro de 1 a 12 que é o mês da venda
  + Recebe um tipo Produto que é um código de produto.
  + Recebe um inteiro que é o numero da filial.
  + Recebe um inteiro 0 ou 1 que é o tipo de venda, normal ou promoção, respectivamente.
* float quantidadeFaturada (CatalogoFaturacao catFact, int mes , Produto p , int filial , int np);
  + Função que dados um mês, um produto, uma filial e um tipo de venda, calcula o valor facturado com esse produto.
  + Recebe um tipo CatalogoFaturacao.
  + Recebe um inteiro de 1 a 12 que é o mês da venda.
  + Recebe um tipo Produto que é um código de produto.
  + Recebe um inteiro que é o numero da filial.
  + Recebe um inteiro 0 ou 1 que é o tipo de venda, normal ou promoção, respectivamente.
  + Retorna um float que é o valor total facturado com tal produto.
* void totalIntervalo (CatalogoFaturacao catFact, int mesi, int mess, int nfiliais , int\* totalvendas , float\* totalfaturado);
  + Função que dado um mês inicial e um mês final, o numero de filiais e dois apontadores, calcula o total facturado e o total de artugos vendidos durante esse período.
  + Recebe um tipo CatalogoFaturacao.
  + Recebe um inteiro que é o mês inicial.
  + Recebe um inteiro que é o mês final.
  + Recebe um apontador para um inteiro que é o resultado do total de artigos vendidos durante o período seleccionado.
  + Recebe um apontador para um float que é o resultado do total facturado durante o período seleccionado.
* ListaGenerica produtosNinguemComprou (CatalogoProdutos catProd, CatalogoFaturacao catFact , int nfiliais, ListaGenerica lg);
  + Função que retorna uma lista com os códigos dos produtos que ninguém comprou.
  + Recebe um tipo CatalogoProdutos que é uma cópia do catalogo de produtos existente no sistema.
  + Recebe um tipo CatalogoFaturacao.
  + Recebe um inteiro que é o numero de filiais do sistema.
  + Recebe um tipo ListaGenerica (ver UTILS)
  + Retorna um tipo ListaGenerica que é uma lista com os códigos de produtos que ninguém comprou.

**Modulo Filiais**

O módulo de Filiais caracteriza-se por uma API com tipos de dados encapsulados onde constam as funções responsáveis por criar, actualizar e consultar a informação referente ás compras efectuadas pelos clientes em cada filial da empresa.

A estrutura de dados principal divide-se em duas partes distintas, distribuídas pelos ficheiros .API-Filiais.C e API-Filiais.h e assentam numa AVL balanceada, por questões de eficiência. (ver AVL).

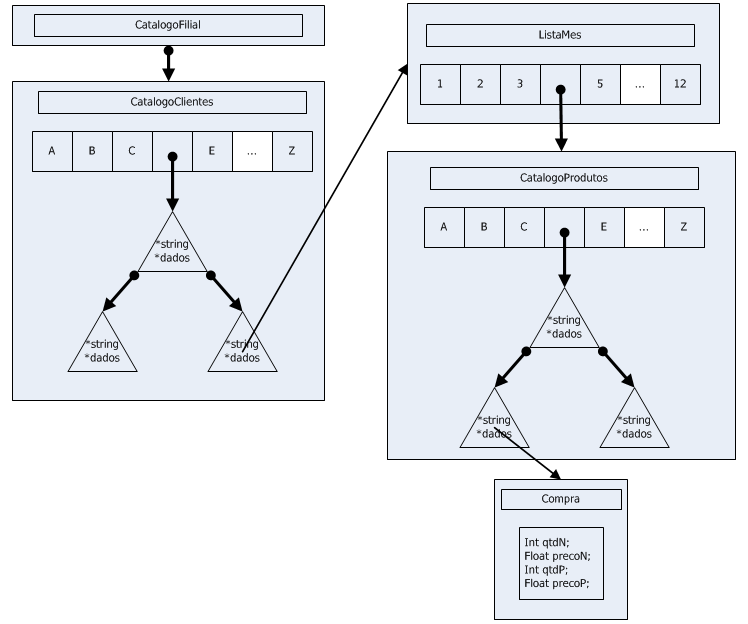
No ficheiro API-Filiais.C temos as seguintes estrutura de dados:

|  |
| --- |
| #define FMTAM 12  #define FCTAM 26  #define MINCLIENTE 1000  #define MAXCLIENTE 5000  #define MINPRODUTO 1000  #define MAXPRODUTO 1999  typedef struct ListaMes{  CatalogoProdutos catMes[FMTAM];  } \*listaMes;  typedef struct ListaFilial{  CatalogoClientes cliente;  int comprasValidas;  } lf1;    typedef struct Compra{  int qtdN;  float precoN;  int qtdP;  float precoP;  } \*compra;  typedef struct ListaDecrescente{  char\* string;  int qtd;  int tamanho;  }\*ld1; |

A estrutura ListaFilial é composta por um catalogo de clientes que é uma copia do catalogo de clientes existes no sistema. Contém um inteiro comprasValidas que é o total de compras validas efectuadas na filial.

A estrutura ListaMes é composta por doze catálogos de produtos, um por cada mês do ano. Esta estrutura é carregada como dado auxiliar de cada cliente do catálogo de clientes representado na Estrutura ListaFilial.(ver modulo Clientes e AVL).

A estrutura Compra é um dado auxiliar, carregado dentro de cada produto onde se inserem as quantidades vendidas e os preços, quer em modo normal, quer em modo promoção.(ver modulo Produtos e AVL)



No ficheiro API-Faturacao.h temos a seguinte estrutura de dados:

|  |
| --- |
| typedef struct ListaFilial \*CatalogoFilial;  typedef struct ListaDecrescente \*listaDecrescente; |

CatalogoFilial é uma estrutura baseada em AVL que contem as compras válidas existentes no sistema em cada filial. No fundo é uma forma de podermos aceder à struct ListaFilial sem que o utilizador de aperceba de como os dados estão guardados, garantindo assim o excapsulamento do tipo de dados.

Da API de Filiais fazem parte as seguintes funções:

* CatalogoFilial iniciaCatFilial(CatalogoClientes catCli);
  + Função responsável por criar um catalogo de compras de uma filial.
  + Recebe um tipo CatalogoClientes que é uma copia do catalogo de clientes existente no sistema.
  + Retorna um tipo CatalogoFilial.
* CatalogoFilial insereVendaFilial(CatalogoFilial catFil, Cliente c, Produto p, int qtd, float preco, int mes, char tipo);
  + Função responsável por inserir um compra valida numa filial.
  + Recebe um tipo CatalogoFilial
  + Recebe um tipo Cliente que é um código de cliente.
  + Recebe um tipo Produto que é um código de produto.
  + Recebe um inteiro que é quantidade comprada.
  + Recebe um float que é um preço de compra.
  + Recebe um inteiro de 1 a 12 que é o mês em que foi efectuada a compra.
  + Recebe um caracter que pode ser N ou P, consoante a compra tenha sido efectuada em modo Normal ou em Promoção
  + Retorna um tipo CatalogoFilial.
* void removeCatFilial(CatalogoFilial catFil);
  + Função que elimina um catalogo de uma filial e faz free() do tipo CatalogoFilial.
  + Recebe um tipo CatalogoFilial.
* int totalCompras(CatalogoFilial catFil);
  + Função que calcula o total de artigos vendido na filial.
  + Recebe um tipo CatalogoFilial.
  + Retorna um inteiro que é o numero de artigos vendidos na filial.
* void clientesContemProduto (CatalogoClientes catCli, CatalogoFilial catFil , Produto p, ListaGenerica genN, ListaGenerica genP);
  + função que escreve na estrutura ListaGenerica os codigos dos clientes que compraram um produto numa dada filial
  + recebe um catalogo de clientes, um catalogo de filiais, o codigo do produto e duas ListaGenerica, uma para os clientes que compraram em regime normal e outra para os clientes que compraram em promoção

**Modulo Ficheiros**

O módulo de Ficheiros caracteriza-se por uma API responsável por manipular os ficheiros onde são armazenados todos os dados refentes à aplicação. Este trata de abrir os ficheiros e carregar os ficheiros, validar os dados e carregar os mesmos nos respectivos módulos.

Da API de Ficheiros fazem parte as seguintes funções:

* CatalogoClientes abreFicheiroClientes(CatalogoClientes catCli);
  + Função responsável pela abertura do ficheiro que contem os códigos de clientes.
  + Recebe um tipo CatalogoClientes.
  + Retorna um tipo CatalogoClientes.
* CatalogoProdutos abreFicheiroProdutos(CatalogoProdutos catProd);
  + Função responsável pela abertura do ficheiro que contem os códigos de Produtos.
  + Recebe um tipo CatalogoProdutos.
  + Retorna um tipo CatalogoProdutos.
* void abreFicheiroVendas(CatalogoFaturacao catFact, CatalogoFilial \*catFil, CatalogoClientes catCli, CatalogoProdutos catProd);
  + Função responsável pela abertura de validação do ficheiro de vendas e pelo carregamento do mesmo no catálogo de facturação e nos catálogos de filiais.
  + Recebe um tipo CatalogoFaturacao
  + Recebe um apontador para um tipo CatalogoFilial.
  + Recebe um tipo CatalogoClientes.
  + Recebe um tipo CatalogoProdutos.

**Modulo Impressão**

O modulo impressão é uma “extensão” do modulo main, que apenas trata da impressão de strings para o ecra. O tamanho de cada pagina esta limitado a 35 codigos por pagina.

Este modulo apenas contem uma função:

* void imprimeString (char \*\* cat, int size);
  + função que imprime para o ecra um array de strings
  + recebe um array de strings e o seu tamanho

**Modulo Utils**

O módulo Utils caracteriza-se por uma API de uso genérico, onde são definidas funções de uso generalizado da aplicação e das suas API.

No ficheiro API-Utils.C temos a seguinte estrutura de dados:

|  |
| --- |
| typedef struct ListaGen{  char\*\* lista;  int tamanho;  int tamMax;  } \*listaGen; |

A estrutura ListaGen é composta por uma lista que é um array de strings, um inteiro tamanho que é o numero de strings que compõe a lista e um inteiro tamMAx que é o tamanho máximo alocado para a lista.

No ficheiro API-Utils.h temos as seguintes estrutura de dados:

|  |
| --- |
| typedef enum {FALSE,TRUE} Boolean;  typedef struct ListaGen \*ListaGenerica; |

Boolean é um tipo de dados enumerado que pode assumir dois valores distinto, TRUE ou FALSE.

ListaGenerica é uma estrutura baseada em arrays de strings. No fundo é uma forma de podermos aceder à struct ListaGen sem que o utilizador de aperceba de como os dados estão guardados, garantindo assim o encapsulamento do tipo de dados.

Da API de Utils fazem parte as seguintes funções:

* Boolean verificaAlpha(char caracter);
  + Função que verifica se um caracter está entre A e Z.
  + Recebe um caracter.
  + Retorna um tipo booleano.
* Boolean verificaNumero(char \*num, int min, int max);
  + Função que verifica se uma string é um numero e se o numero se encontra entre um mínimo e um máximo.
  + Recebe uma string
  + Recebe um inteiro que é o mínimo.
  + Recebe um inteiro que é o maximo.
  + Retorna um tipo booleano.
* ListaGenerica initListaGen(int n);
  + Função que inicia uma lista genérica de strings vazia.
  + Recebe um inteiro que é o tamanho máximo da lista genérica.
  + Retorna um tipo ListaGenerica.
* ListaGenerica insereListaGen(ListaGenerica lg, char\* string);
  + Função que insere uma string numa lista genérica de strings
  + Recebe um tipo ListaGenerica.
  + Recebe uma string.
  + Retorna um tipo ListaGenerica.
* int retornaPosListaGen(ListaGenerica lg);
  + Função que retorna o numero de elementos existentes na lista e por consequência a próxima posição a inserir elementos.
  + Recebe um tipo ListaGenerica.
  + Retorna um inteiro que é a próxima posição onde inserir elementos e também o numero de elementos existentes na lista.
* int retornaTamMaxListaGen(ListaGenerica lg);
  + Função que retorna o tamanho maximo da Lista de strings
  + Recebe um tipo ListaGenérica
  + Retorna um inteiro que é o numero máximo de strings que a lista pode conter.
* char\*\* retornaListaGen(ListaGenerica lg);
  + Função que retorna o apontador para um arrays de strings
  + Recebe um tipo ListaGenerica.
  + Retorna um apontador para um array de strings.
* void removeListaGen(ListaGenerica lg);
  + Função que elimina uma lista genérica e faz free() do tipo ListaGenerica.
  + Recebe um tipo ListaGenerica.

**Modulo AVL**

No modulo avl-strings encontra-se a implementação das avl usadas neste preojeto.

As avl usadas tem a seguinte extrutura de dados:

|  |
| --- |
| typedef struct btree {  int balance;  char \*string;  void \*dados;  struct btree \*left, \*right;  }Node; |

A estrutura Node e composta pelo fator de balanceamento int balance que gere o balanceamento das arvores;

char \*string, funciona como fator de comparação e armazena os codigos de cliente ou produto;

void \*dados, que armazena qualquer estrutura que lhe seja atribuida;

struct btree \*left, \*right, que armazenam os apontadores para as árvores da direita e da esquerda;

Do modulo avl-strings fazem parte as seguintes funções:

* BTree insertAVL(BTree t, char \*str, int \*grow, void \*dados);
  + função que inicia a inserção de um elemento na árvore
  + recebe a árvore na qual vai ser inserido o elemento ,a string que vai ser inserida, o fator de balanceamento e os dados genericos a inserir
  + retorna o apontador para a árvore com o elemento
* BTree insertRight(BTree t, char\* str, int \*grow, void \*dados);
  + função que insere um elemento a direita de um nodo
  + recebe a árvore na qual vai ser inserido o elemento ,a string que vai ser inserida, o fator de balanceamento e os dados genericos a inserir
  + retorna o apontador para a árvore
* BTree insertLeft(BTree t, char\* str, int \*grow, void \*dados);
  + função que insere um elemento a esquerda de um nodo
  + recebe a árvore na qual vai ser inserido o elemento ,a string que vai ser inserida, o fator de balanceamento e os dados genericos a inserir
  + retorna o apontador para a árvore
* BTree balanceRight(BTree t);
  + função que corrige o balanceamento da arvore a direita, efetuando todas as rotações necessárias
  + recebe o apontador para a arvore
  + retorna o apontador para a arvore já balanceada
* BTree balanceLeft(BTree t);
  + função que corrige o balanceamento da arvore a direita, efetuando todas as rotações necessárias
  + recebe o apontador para a arvore
  + retorna o apontador para a arvore já balanceada
* BTree rotateRight(BTree t);
  + função que efetua uma rotação simples a direita
  + recebe o apontador para a arvore
  + retorna o apontador para a arvore depois da rotação estar concluida
* BTree rotateLeft(BTree t);
  + função que efetua uma rotação simples a esquerda
  + recebe o apontador para a arvore
  + retorna o apontador para a arvore depois da rotação estar concluida
* int exists (BTree t, char \*str);
  + função que efetua uma procura na avl
  + recebe o apontador para a árvore e a string que se quer encontrar
  + Se a string existir na avl retorna 1, se não existir retorna 0
* void\* retornaDados (BTree t, char \*str);
  + função que retorna tudo que esta armazenado no campo void \*dados de um nodo da avl
  + recebe o apontador para a arvore e a string que identifica o nodo onde se encontra o que se pertende retornar
  + retorna o que quer que seja que esteja armazenado no nodo da arvore
* BTree deleteAvl (BTree t);
  + função que remove e faz free a uma avl
  + recebe o apontador para a árvore que se quer eliminar
  + reotorna o apontador para a arvore depois de removida, NULL
* char\*\* criaArrayString(BTree catalogo, int tamStrint, int tamAVL);
  + função que cria um array de strings apartir de uma avl
  + recebe o apontador para a avl, o tamanho dos elementos a inserir na string e o tamanho da avl
  + retorna o array de strings
* BTree avlCopy(BTree t);
  + função que cria uma uma copia de uma avl
  + recebe o apontador para a árvore
  + retorna o apontador para a copia da árvore
* void insereDados(BTree t, char \*str, void\* dados);
  + função que insere dados no campo void \*dados de um nodo
  + recebe o apontador para a arvore, a string que identifica o nodo e os dados a introduzir
* char\* retornaString (BTree t);
  + função que retorna a string de um nodo de uma arvore
  + recebe o apontador para o nodo da arvore
  + retorna a string contida nesse nodo
* void avlInorder (BTree t, char\*\* str , int \*j);
  + função que escreve num array todas as strings que a avl contem
  + recebe um apontador para a árvore, o array para onde vão ser copiadas as strings, e o indice do array

**Decisões de Navegação sobre estruturas**

**Testes de Performance**

|  |  |  |
| --- | --- | --- |
|  | Vendas\_1M | Vendas\_5M |
| carregar dados | 5.607 segundos | 24.5678 segundos |
| Query 2 dada a letra 'A' | 0.0023 | 0.0023 |
| Query 3 valores totais no mês 6, produto AF1184 | 0.000119 | 0.000115 |
| Query 4 | 1.8458 | 1.8517 |
| Query 5 cliente Z5000 | 0.000100 | 0.000234 |
| Query 6 do mês 1 ate ao mês 12 | 1.6269 | 1.8954 |
| Query 8, produto AF1184 | 0.0638 | 0.0614 |

**Makefile e gráficos de Dependencias**

**Conclusão**