Universidade do Minho

Relatório – Laboratórios de Informática III

MIEI-2º Ano- 2º semestre

Gestão de Vendas

Grupo 40

Catarina Gil
A85266

Margarida Campos
A85166

Tânia Rocha
A85176

ÍNDICE

1. Introdução

Este projeto foi proposto pelos docentes da unidade curricular Laboratórios de Informática III e tem como principal objetivo a realização de um programa que
faça a gestão de vendas.

Neste trabalho foi nos solicitado o desenvolvimento e a aplicação de grandes quantidades de dados, assim como a sua eficácia.

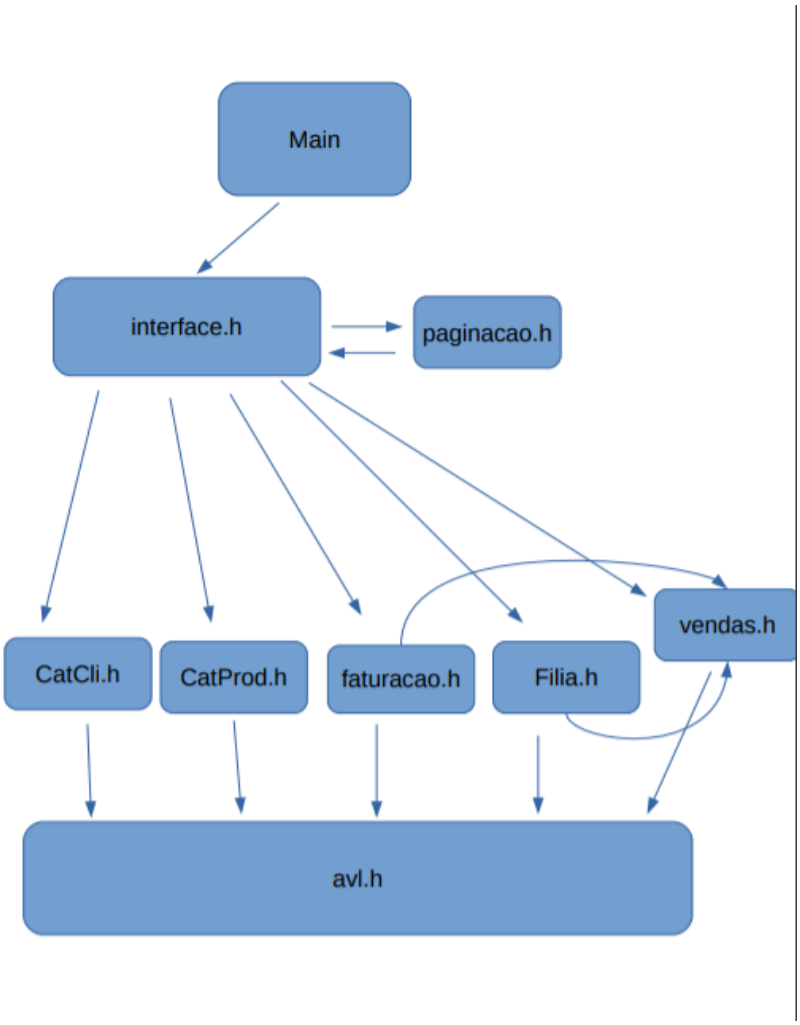


Gráfico de dependências

2. Descrição dos Módulos

2.1. AVL's

```
struct avl {
    char* code;
    int height;
    void* info;
    struct avl *left, *right;
};


struct myAvl {
    int total;
    Avl avl;
};
```

Avl : Nodo constituído por um char* referente a um código , uma altura, apontador para uma estrutura e apontadores para os nodos da direita e esquerda.

MY_AVL: Árvore constituída por Avl's e o seu respetivo tamanho.

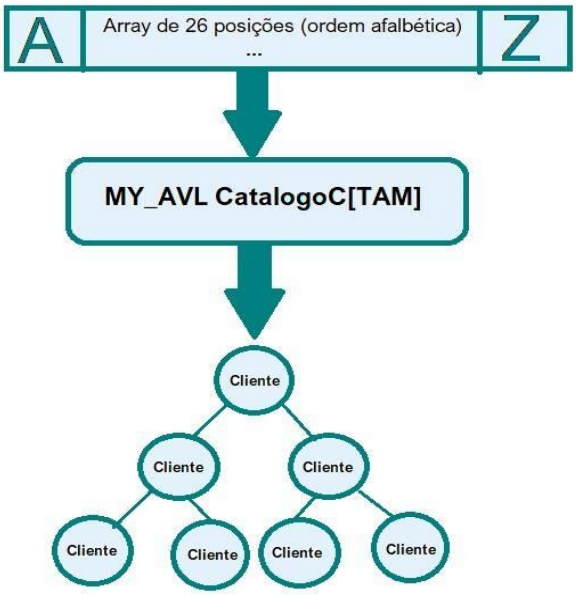
```
typedef struct avl *Avl;

typedef struct myAvl *MY_AVL;
```

2.2. Catálogo Clientes/Produtos

Foi utilizado o mesmo método para os Catálogos dos Clientes e dos Produtos, isto é, um array com 26 MY_AVL's correspondente à letra inicial do código referentes ao cliente/produto, assim como foram criadas funções com o mesmo propósito.

- **CATCLI initCatCli() :** inicializa o Catálogo de Clientes;
- **int validaClientes (char* cliente) :** verifica se um Cliente é válido;
- **CATCLI insereCli(CATCLI catClientes, char cliente[]) :** insere um cliente no Catálogo;

- **BOOL jaExisteC (CATCLI catClientes, char cliente[]) :** verifica se já existe um dado cliente no Catálogo;
- **int contaCliLetra (CATCLI catClientes,char letra) :** conta o número de clientes que existem numa árvore (correspondente à sua letra inicial) ;
- **int contaCliCat(CATCLI catClientes) :** conta o número de clientes no Catálogo;
- **void printCLI(CATCLI clientela)** : imprime Catálogo de Clientes (utilizado para testes);




```c
struct catCli{
    MY_AVL CatalogoC[TAM];
};
```

```c
typedef struct catCli *CATCLI;
```

2.3. Vendas

```c
struct venda{
  char* cliente;
  char* produto;
  double preco;
  int quant;
  char* tipo;
  int filial;
  int mes;
  char* vendaCompleta;
};

struct catVenda{
    MY_AVL  catvendas;
    };
```

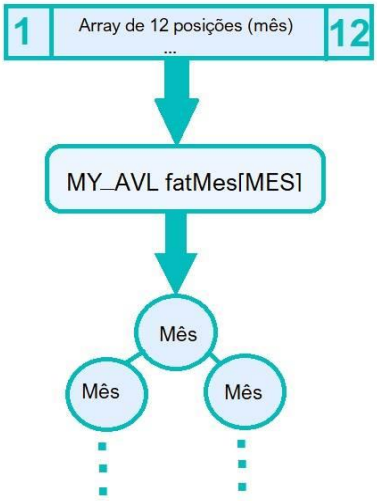
```c
typedef struct venda *VENDA;
typedef struct catVenda *CATVENDA;
```

• **VENDA criaVenda() :** inicialização da estrutura Venda;
• **CATVENDA inicializaVendaDiv():** inicializa uma venda dividida;
• **char** tokenizeLinhaVendaDyn(char* vendaRaw):** divide a linha de uma venda pelos seus componentes (cliente,produto,preço,filial,quantidade,mes,tipo);
• **VENDA mkVendaStruct(char* linhaVenda):** após a divisão da linha de venda atribui o conteúdo divido a cada componente da struct VENDA;
• **CATVENDA insereVEND (CATVENDA catV, char* linha):** insere uma dada venda no Catálogo de Vendas;
• **int validaVendas (CATCLI clientes,CATALOGO_PRODUTOS produtos, char* vendida**): verifica se uma venda é válida;
• **char* procuraProduto(CATALOGO_PRODUTOS catProd,VENDA v,int tam):** vê se um produto nao existe,conta os que não existem e mete os num array;
• **int numProdNCompra(CATALOGO_PRODUTOS catProd,CATVENDA v):** número de produtos válidos que nunca são comprados;
• **int numCliNCompra(CATCLI catCli,CATVENDA v):** número de clientes válidos que não fazem compras;
• **int treeTraversal(int a, Avl nod, char* arr[]):**Função que armazena num array todos os elementos dos nodos de uma AVL.
• **int transformaEmvenda(int a,CATVENDA ardeu,Avl v):**Armazena num tipo catalogo apenas os produtos das vendas de um catalogo de vendas;
• **int transformaEmvendaCLI(int a,CATVENDA ardeu,Avl v**): Armazena num tipo catalogo apenas os clientes das vendas de um catalogo de vendas;
• **int firstTransversal(MY_AVL e,char* array[]):**Esta é uma função utilizada para chamar uma outa função mas com o modulo Avl em vez de MY_AVL;
• **int travessiaArrayAux(int u,Avl e,Avl nod, char* arr**[]):Função que armazena num array todos os elementos dos nodos de uma AVL, dependendo da condição de os elementos desta estarem presentes numa outra avl;

2.4. Faturação

Estrutura: Array com 12 MY_AVL's correspondente a cada mês.

- **FATURACAO iniciaFat ():** inicialização da Faturação;
- **int insereFaturacao (FATURACAO faturacao,char* charvenda,char* tt) :** insere o código de uma venda na estrutura Faturação;
- **void AddToArray(Avl node, char* arr[], int *i) :** insere nodos das árvores constituintes da Faturação num array;
- **int faturMes (char* arr[],int tam, int nVN, int nVP,float fVN,float fVP) :** calcula a faturação de um determinado mês.

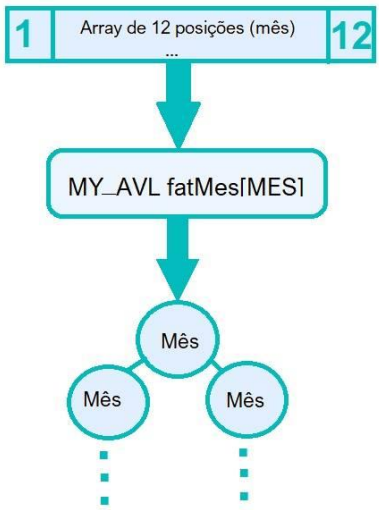



```
struct faturacao{
    MY_AVL fatMES [MES];
};
```

```
typedef struct faturacao *FATURACAO
```

2.5. Filial

Estrutura: Array com 3 MY_AVL's correspondente a cada filial.




```
struct Filial{
  MY_AVL fil[F];
};
```

```
typedef struct Filial *FILIAL;
```

--

- **FILIAL iniciaFilial() :** inicializa a Filial;
- **int insereFilial (FILIAL filial,char* charvenda,char*ttt) :** insere código das vendas na estrutura da Filial;
- **void AddToArrayF(Avl node, char* arr[], int *i)** : insere nodos das árvores constituintes da Filial num array;
- **int travessiaArrayFILIAL(int u,Avl f1,Avl f2, Avl f3,char* arr[]) :** adiciona a um array os clientes que compram nas tres filiais;

2.6. Makefile

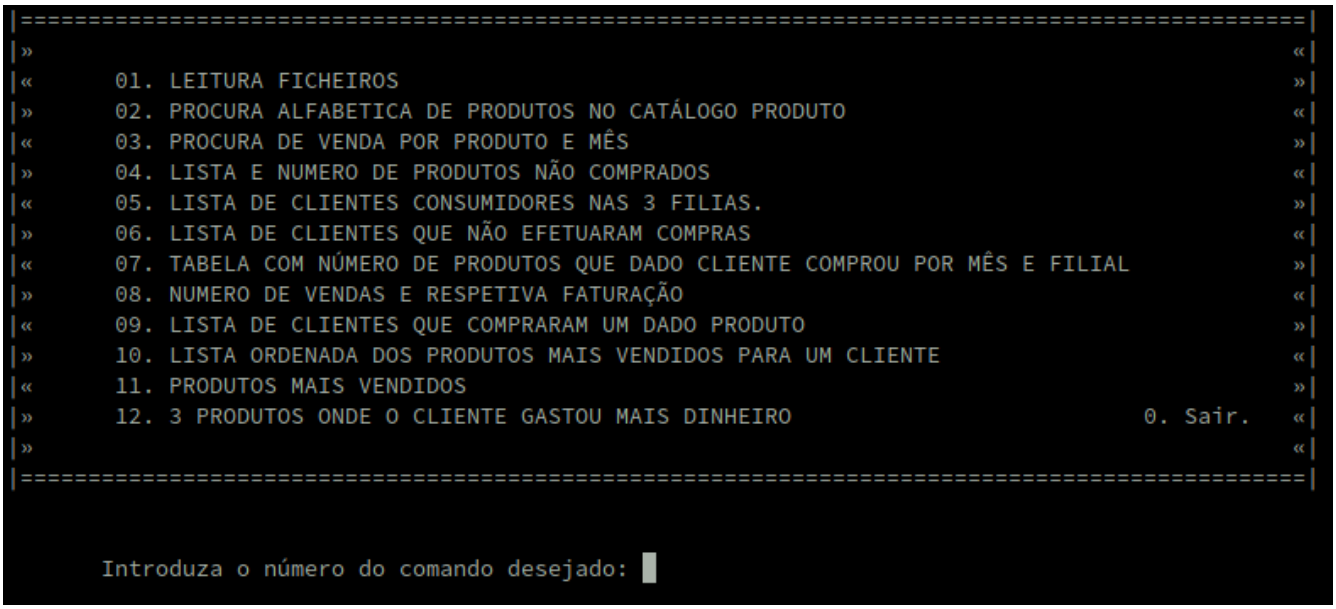
```
CFLAGS =-g -Wall -O2
PROJECT=mainP

$(PROJECT): main.o CatProd.o CatCli.o avl.o faturacao.o interface.o vendas.o paginacao.o Filial.o
    gcc -o $(PROJECT) main.o CatProd.o CatCli.o avl.o faturacao.o interface.o vendas.o paginacao.o Filial.o

clean:
    rm -rf *.o $(PROJECT)

main.o: main.c main.h CatCli.h CatProd.h avl.h faturacao.h interface.h vendas.h paginacao.h Filial.h
CatProd.o: CatProd.c CatProd.h main.h avl.h CatCli.h faturacao.h interface.h vendas.h paginacao.h Filial.h
CatCli.o: CatCli.c CatCli.h main.h CatProd.h avl.h faturacao.h interface.h vendas.h paginacao.h Filial.h
avl.o: avl.c avl.h CatProd.h main.h CatCli.h faturacao.h interface.h vendas.h paginacao.h Filial.h
faturacao.o: faturacao.c faturacao.h avl.h CatProd.h main.h CatCli.h interface.h vendas.h paginacao.h Filial.h
interface.o: interface.c interface.h faturacao.h avl.h CatProd.h main.h CatCli.h vendas.h paginacao.h Filial.h
vendas.o: vendas.c vendas.h interface.h faturacao.h avl.h CatProd.h main.h CatCli.h paginacao.h Filial.h
paginacao.o:paginacao.c paginacao.h vendas.h interface.h faturacao.h avl.h CatProd.h main.h CatCli.h Filial.h
Filial.o:Filial.c paginacao.h vendas.h interface.h faturacao.h avl.h CatProd.h main.h CatCli.h Filial.h
```

2.7. Main

Na main inicializamos todas as structs criadas, ou sejas, os Catálogos dos Clientes e dos Produtos,as Vendas, a Faturaçao e a Filial, passando-as como argumentos na função interface.

```
int main(int argc, char** argv){
    CATCLI catClients = initCatCli();
    CATVENDA vendas = inicializaVendaDiv();
    CATALOGO_PRODUTOS p = iniciaProdutos();
    FATURACAO fact = iniciaFat();
    FILIAL filial = iniciaFilial();
    interface(catClients,p,vendas,fact,filial);

    return 0;

}
```

2.8. Interface



Para aceder cada querie basta digitar o número correspondente à mesma.

```c
printf("Carregue qualquer numero para sair.");
int k;
scanf("%d",&k);
if(k==0) interface(cli, prod,vendas,fact,filial);
else (interface(cli,prod,vendas,fact,filial));
return 0;
```

Para cada querie utilizamos estas linhas de código para voltar ao menu principal.

3. Paginação

A paginação é um módulo criado com o objetivo de imprimir listadamente um array com os elementos desejados, ou seja, aquando a necessidade de imprimir elementos de uma querie, estes são armazenados num array criado com o intuito de este ser utilizado para imprimir ordenadamente e de maneira organizada.

Neste módulo, é também feita a contabilização dos elementos a imprimir e a contagem das páginas necessárias, como também um método de regresso ao menu principal.

4.Conclusão

Concluindo, este trabalho ajudou-nos a pensar na eficiência e a criar estruturas que nos ajudaram a lidar com a leitura de uma grande capacidade de informação pedida pelo utilizador.