Universidade Federal de Viçosa

CAMPUS FLORESTAL

SAV

SISTEMA AUTOMATIZADO DE VOOS

Pablo Miranda Batista	3482
Lucas Ranieri Oliveira Martins	3479
João Victor Magalhães Souza	3483

FLORESTAL – MG
SETEMBRO DE 2018

Sumário

1. Objetivo e Inicialização do projeto

O objetivo do trabalho foi criar um sistema de aeroporto, com os voos de um dia organizado a partir de uma matriz de horário. Quanto à intensão acadêmica, ela resultou em uma interiorização do trabalho em equipe dos membros, onde foi desenvolvido compromisso e colaboração.

Inicialmente, o projeto teve uma divisão de tarefas, onde o aluno Pablo ficou responsável por fazer as TAD's Voo e lista encadeada, Lucas com as TAD's Item Matriz e Matriz e, por fim, João com o Main e auxílio ao Lucas no desenvolvimento da Matriz. Porém durante o desenvolvimento do projeto todas as partes se envolveram no desenvolvimento em todas as etapas.

O projeto do trabalho prático iniciou tendo uma divisão bem específica de tarefas entre os membros do grupo, onde cada membro buscava fazer uma parte, sem ter uma base das partes dos outros membros, buscamos maior otimização do tempo. Porém, nos primeiros dias esse método se mostrou ineficaz, pois a partir do tipo abstrato de dado referente ao "item matriz" cada parte do projeto tinha necessidade do conhecimento pelo menos básico das parcelas feitas pelos outros membros. Nossa dificuldade inicial quanto à divisão foi resolvida, criando uma pasta compartilhada, onde tivemos maior interatividade entre cada membro e todas as partes do trabalho, mesmo que algum membro fosse o maior responsável por alguma delas.

2. Módulos desenvolvidos

I. Pablo Miranda Batista – TAD Voo e TAD Lista

Inicialmente foi criada a estrutura de dados principal, a TAD Voo, onde ficariam armazenados todos os dados do Voo solicitados pelo cliente,

```
//Estrutura da TAD voo
typedef struct {
    int VID, pista;
    char aerodecolar[50], aeropousar[50], horadecolar[10], horapouso[10];
}tipovoo;
```

como pode se ver na imagem acima, optamos por armazenar a hora de decolagem, hora de pouso e os aeroportos de decolagem e pouso em variáveis do tipo char, pois assim iríamos ter uma variedade de opções de preenchimento das mesmas. Já o ID e a pista foram optados uso do tipo int, pois só iria armazenar neles somente números.

Já as funções referentes à estrutura TAD Voo se limitaram somente às padrões, que seriam as funções de inicializar novo voo, atribuição (responsável por preencher o voo com as informações digitadas pelo usuário), e as funções “GET” (retorna uma única informação da estrutura) e “SET” (altera um dos campos da estrutura).

Após o término da criação e teste da TAD Voo, foi criado a TAD Lista Voos, responsável por criar e administrar listas encadeadas de voos. Para essa parte, desenvolveu-se a seguinte estrutura,

```
//Estrutura da celula da lista encadeada
typedef  struct celula *apontador;
typedef struct celula{
    tipovoo voo;
    struct celula* prox;
} tcelula;

// Estrutura da lista encadeada
typedef struct {
    tcelula* primeiro;
    tcelula* ultimo;
} ListaVoos;
```

essa estrutura é bem simples porém bem eficiente. Podemos ver que dentro de cada célula da lista existe uma estrutura voo, como visto anteriormente, e um ponteiro, que ira armazenar o endereço da próxima célula da lista.

Como esse TAD é o núcleo de todo o sistema, foram implementadas funções de extrema importância, pois se essa TAD não for eficaz, o sistema todo será ineficiente. Entre as funções criadas se destacam as funções:

Lista_insere

```
void Lista_insere (ListaVoos* lista, tipovoo* voo){
    tcelula *voo_atual, *proxvoo, *novo;

    novo = (apontador) malloc(sizeof(tcelula));
    novo->voo = *voo;

    voo_atual = lista->primeiro;
    proxvoo = lista->primeiro->prox;

    while (proxvoo != NULL) {
        if (strcmp(proxvoo->voo.horadecolar, voo->horadecolar) >= 0) {
            break;
        }
        voo_atual = proxvoo;
        proxvoo = proxvoo->prox;
    }
    novo->prox = proxvoo;
    voo_atual->prox = novo;
    if (lista->ultimo->prox != NULL){  //este IF esta conferindo se a ne
        lista->ultimo = lista->ultimo->prox;
        lista->ultimo->prox = NULL;
    }
}
```

Essa é a função “insere” responsável por inserir um novo voo na posição anterior ao voo já existente na lista com hora de colagem mais similar ao do voo a ser incluído. Para que isso aconteça essa função inicialmente reserva no HEAP uma quantidade de memoria suficiente para armazenar uma célula da lista encadeada, depois ela percorre a lista procurando o voo com hora de decolagem maior, porém similar ao do novo voo. Se não houver nada na lista, será inserido na primeira posição da lista, se a hora de decolagem do novo voo for maior que de todos os voos já existentes na lista, ele será inserido na ultima posição da lista.

Lista_remove

```
int Lista_remove (ListaVoos* lista, tipovoo *voo, int vid){
    tcelula *voo_atual, *prox_voo;

    voo_atual = lista->primeiro;
    prox_voo = lista->primeiro->prox;

    while(prox_voo != NULL && prox_voo->voo.VID != vid){
        voo_atual = prox_voo;
        prox_voo = voo_atual->prox;
    }

    if(prox_voo != NULL){
        *voo = prox_voo->voo;
        voo_atual->prox = prox_voo->prox;
        free(prox_voo);
        if(lista->ultimo->prox != NULL){
            lista->ultimo = lista->ultimo->prox;
            lista->ultimo->prox = NULL;
        }
        return 1;
    }
    return 0;
}
```

Essa função tem a tarefa de remover um voo da lista, utilizando o ID digitado pelo usuário para encontra-lo. Para que essa função fosse funcional foi necessária à criação de dois ponteiros auxiliares, que serão responsáveis por apontar para as células da lista enquanto a percorremos, assim eliminamos o risco de ter alguma perda de endereço. Enquanto a lista é percorrida, a função está acessando o campo ID de cada célula para ver se esse e igual ao ID solicitado pelo usuário, quando ele encontrar o voo com ID igual ele ira fazer com que a célula anterior ao voo a ser removido aponte para o voo seguinte ao voo a ser removido, feito isto a função salva as informações do voo a ser removido em um voo que existe fora da função e depois desloca o espaço da memoria onde este armazenado o voo usando o comando *“free”.*

II. Lucas – TAD Item Matriz e Matriz

Em geral, foi seguido o roteiro do trabalho, onde foi criada a estrutura do Item matriz, onde existe uma lista de voos e duas variáveis de auxilio que armazenarão a quantidade de voos que existe na lista e a hora da ultima modificação feita na lista.

```c
// definição do tipo celula da matriz
typedef struct{
    ListaVoos lvoo;
    int n_voos;
    char h_ultima_att[10]; //ex: 00:00:00
}Item_Matriz;
```

Já nas funções, foram implementadas as padrões de inicialização, alteração (SET's) e leitura (GET's). Posteriormente, foi criada uma função de conversão do tipo char para int, para quando necessário o uso da hora de decolagem e hora de pouso como coordenadas de onde ir na matriz.

```c
//recebe um char
//e retorna a conversao das dois primeiros caracteres convertidos em um int
//ex: char = 11:00:00 retorna int = 11
int conversor_ij (char* hora){
    int n = ((hora[0] - '0')*10) + (hora[1] - '0');
    return n;
}
```

Ela pega a primeira posição do char e o transforma em um numero na casa das dezenas e pega o segundo número do char e o converte em um número unitário, para concluir somam os dois números encontrados.

No projeto solicitava que em toda vez que houvesse uma alteração na matriz, para ser modificado a hora armazenada na variável "ultima atualização" para a hora atual. Nessa etapa, usufruímos da biblioteca <time.h>.

```c
void Set_h_ultima_att(Item_Matriz *IM){
    char hora[10];
    _strtime(hora); // essa funcao recebe um char e armazena nele a hora atual do computador
    strcpy(IM->h_ultima_att, hora);
}
```

Utilizamos a função "strtime", que é capaz de pegar a hora do computador no momento que ela foi chamada e transforma-la em um char.

Já a matriz, foi o local do projeto onde todas as TAD's e suas funções interagiam uma com as outras, para que isso fosse possível criamos a seguinte estrutura,

```c
//definiçao do tipo matriz
typedef struct{
    Item_Matriz IMatriz[24][24]; //[partida][chegada]
    char data[10]; //ex: 20/20/2000
    int n_voos;
}Matriz_Voos;
```

como pode ser visto, a estrutura é basicamente uma matriz 24 por 24 onde cada posição representa uma combinação de hora de decolagem e pouso, um campo para armazenar a data atual, ou seja, o dia que esse sistema esta administrando e um campo que armazena a quantidade total de voos administrados pela matriz.

Existe varias funções que operam nessa matriz, porém elas estão sendo utilizadas para administrar e organizar o uso das funções de todas os TAD's implementados. Basicamente, todas as funções estão percorrendo a matriz com o auxilio de "for" ou "while" e utilizando as funções das demais TAD's para executar os comandos solicitados.

Exemplo:

```c
int remover_voo(Matriz_Voos *MVoos, tipovoo *voo, int Vid){
    int i=0, j=0;

    for (i=0;i<24;i++) {
        for (j=0;j<24;j++) {
            if(ProcuraVoo_lista(&MVoos->IMatriz[i][j].lvoo, voo, Vid) == 1){
                Lista_remove(&MVoos->IMatriz[i][j].lvoo, voo, Vid);
                Set_h_ultima_att(&MVoos->IMatriz[i][j]);
                Set_n_voos(&MVoos->IMatriz[i][j]);
                return 1;
            }
        }
    }
    return 0;
}
```

A função está percorrendo cada posição da matriz com o auxilio do "for", e em cada posição da matriz ele utiliza a função "procura", da TAD lista, para checar o voo a ser removido na posição, se a função retornar "1" ele chama a função "remove" da TAD lista e as funções de "set" da TAD item matriz.

Além disso, a TAD matriz e responsável por imprimir os voos solicitados pelo usuário e verificar se a matriz e esparsa. As funções de imprimir consistem em percorrer a matriz e imprimir os voos por hora de decolagem, hora de pouso ou todos os voos.

III. João – Main.c

Por fim, a conclusão do projeto deu-se como um processo bem delicado. Isso porque nossa divisão em relação ao TP ficou um pouco “distante” entre nós. O desafio foi buscar entender os TADs e organizá-los no final.

A primeira parte do Main (Sistema Interativo) não teve grande mistério; basicamente se constituiu em:

- Criar as variáveis que iriam armazenar as entradas do usuário.
- Montar o Painel Interativo.
- Chamar as funções.

Exemplo*:*

Função b: Inserir um novo voo.

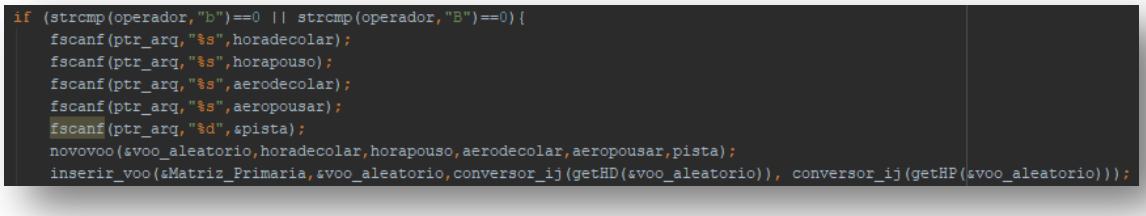
```
if (strcmp(operacao_interativa,"b")==0 || strcmp(operacao_interativa,"B")==0){
    printf("DIGITE A HORA DE DECOLAGEM: ");
    scanf("%s",horadecolar);
    printf("DIGITE A HORA DE POUSO: ");
    scanf("%s",horapouso);
    printf("DIGITE O AEROPORTO DE DECOLAGEM: ");
    scanf("%s",aerodecolar);
    printf("DIGITE O AEROPORTO DE POUSO: ");
    scanf("%s",aeropousar);
    printf("DIGITE A PISTA: ");
    scanf("%d", &pista);
    novovoo(&voo_aleatorio, horadecolar, horapouso, aerodecolar, aeropousar, pista);
    inserir_voo(&Matriz_Primaria,&voo_aleatorio,conversor_ij(getHD(&voo_aleatorio)), conversor_ij(getHP(&voo_aleatorio)));
```

A segunda parte do Main (Sistema por Arquivo) era considerada a mais complicada. Isso porque, particularmente, eu não estava muito confiante em relação ao que havia aprendido sobre manipulação de arquivos. Entretanto, corri atrás sobre como era feita a leitura de arquivo, como armazenar em variáveis e determinar os intervalos de leitura que cada variável deveria aceitar. Função de leitura:

```
while ((fscanf(ptr_arq,"%s",operador))!= EOF){
```

ptr_arq : Ponteiro para *FILE*.
Operador: Variável que armazena qual será a função a ser chamada.
“Enquanto a leitura não identificar EOF, continue a ler”.

Utilizamos um método bem inteligente para a leitura de arquivo. Primeiramente lemos qual é o operador. Em seguida, verificamos quais são os parâmetros que esse operador necessita para chamar a função. Após isso, lemos esses parâmetros e passamos para a função escolhida.



```
if (strcmp(operador,"b")==0 || strcmp(operador,"B")==0){
    fscanf(ptr_arq,"%s",horadecolar);
    fscanf(ptr_arq,"%s",horapouso);
    fscanf(ptr_arq,"%s",aerodecolar);
    fscanf(ptr_arq,"%s",aeropousar);
    fscanf(ptr_arq,"%d",&pista);
    novovoo(&voo_aleatorio,horadecolar,horapouso,aerodecolar,aeropousar,pista);
    inserir_voo(&Matriz_Primaria,&voo_aleatorio,conversor_ij(getHD(&voo_aleatorio)), conversor_ij(getHP(&voo_aleatorio)));
```

Estamos verificando se o operador é o **b** (Inserir novo voo). Caso seja, a função deve ler os parâmetros tais que a função **b** requer. São eles: **Hora de Decolar, Hora de Pouso, Aeroporto de Decolagem, Aeroporto de Pouso e a Pista.** Após isso, como também feito na Operação Interativa, caso o voo seja inserido com sucesso, seu VID é gerado.

3. Conclusão

Este projeto foi uma experiência, pois mostrou que programação não é só jogar os comandos na IDE e compilar. É muito mais que isso, tem que haver planejamento, organização e na administração dos recursos e tempo. A realidade é que sentar e digitar linhas de códigos é a ultima tarefa que devemos fazer em um projeto como esse.

Inicialmente, em nosso trabalho, acreditamos que poderíamos fazer como estávamos acostumados, cada um sentar na frente de seu computador e programar a parte que foi designada a ele, porém com o passar dos dias vimos que não daria certo, pois cada uma das partes era complementar da outra. Então decidimos sentar e conversar sobre um planejamento para desenvolver esse trabalho. No fim, decidimos por todos ficarem envolvidos sempre na mesma TAD, assim todos saberiam como ela funciona, mas sempre houve uma pessoa que estava responsável por passar as ideias para a IDE, na documentação está mostrando quem foi o responsável por fazer essas transferências de ideias para o projeto.

Durante o desenvolvimento tivemos vários problemas, mas entre eles os que geraram mais dificuldade para resolver foram: o manuseio da lista encadeada e a utilização de leitura de arquivo. Sobre a lista, nossos problemas foram devidos à falta de conhecimento, pois ainda não chegara à aula sobre o assunto, mas depois de alguns dias pesquisando, conseguimos fazer funcionar e depois utilizamos a aula sobre lista encadeada para aperfeiçoar as funções. Já sobre a leitura de arquivo, estávamos tendo problemas de fazer a leitura de todos os parâmetros na sequência correta, porém depois de algumas conversas com os monitores, conseguimos resolver o problema.

Em geral não tivemos grandes problemas para concluir o trabalho, mas sim ganhamos uma grande experiência para projetos futuros. No inicio, quando nos foi apresentado esse TP, aparentava que não conseguiríamos desenvolvê-lo, mas agora que o concluímos, vejo que ele não é nada além daquilo que já conhecemos e usamos, com diferença na questão de saber utilizar engenhosamente tais conhecimentos.