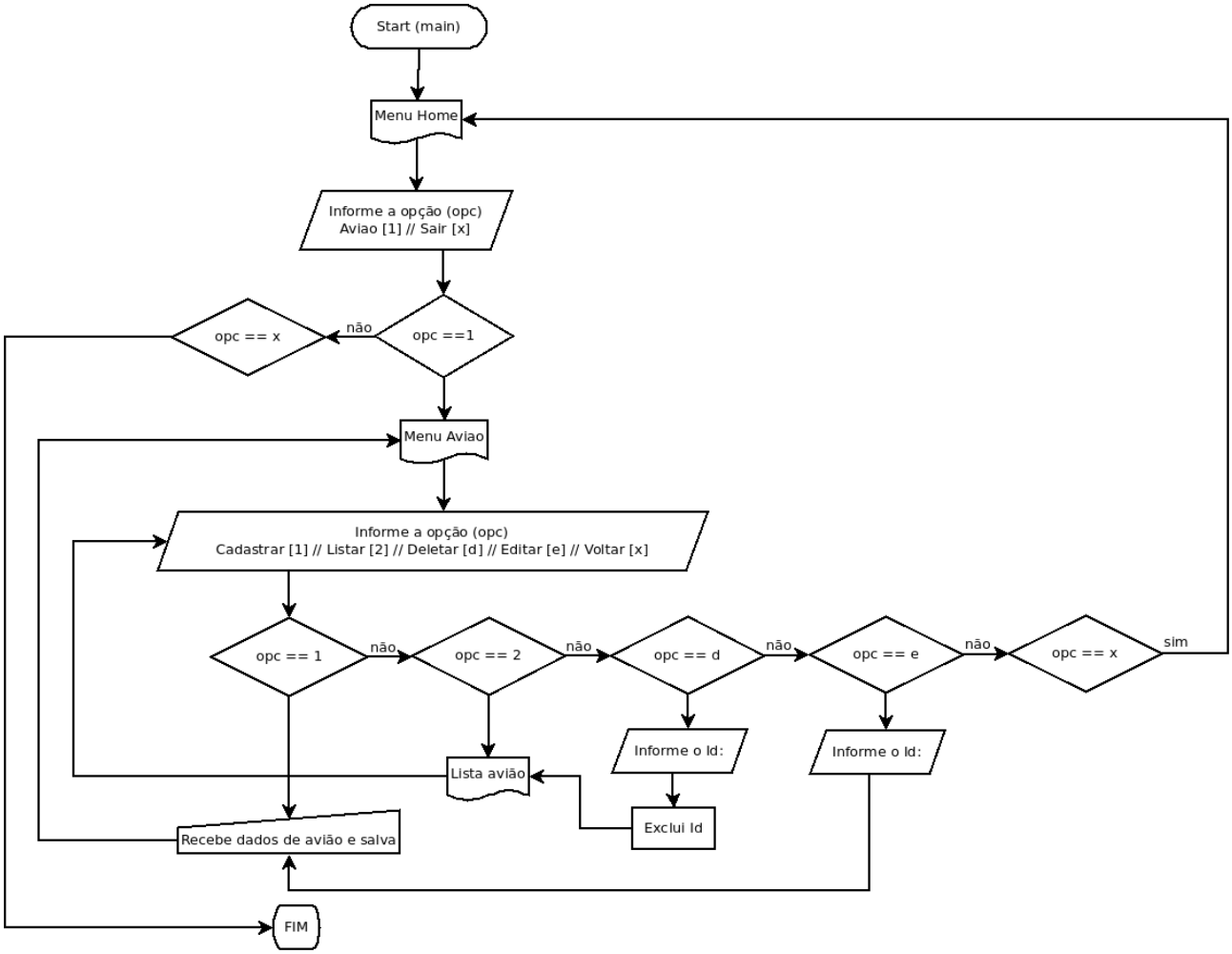
ANHANGUERA
EDUCACIONAL

Desenvolvimento do projeto proposto

Diagrama de sequencia do projeto:



Ferramentas utilizadas

Desenvolvimento do projeto / código:

- Qt Creator
- http://qt.digia.com/
- Suporte a plataformas Windowns e Linux

Diagrama de sequencia:

- Dia
- http://dia-installer.de/

Sistema operacional

- Ubuntu 13 + Gnome3
- http://www.gnome.org/getting-gnome/

Gravação do vídeo de apoio

- Kazam
- https://launchpad.net/kazam

Repositório do projeto

- gitHub
- https://github.com/miamarti/ATPS-cPlusPlus-2s2013

Gerenciador local

- Cola Git GUI
- http://git-cola.github.io/
-

O Projeto

Todo projeto foi desenvolvido em camadas, sendo elas:

- BaseDAO
- Responsável por armazenar todos os dados da aplicação (Struct // Emulação do Banco de Dados)
- BaseDao.h
- Model
- Responsável pela normalização dos dados (Beans)
- AviaoBean.h
- Business
- Responsável pela execução das ações relacionadas ao negócio
- AvioesBO.h
- MenuBO.h
- View
- Responsável por gerenciar construtores de tela
- AvioesView.h
- Helpers
- Responsável por gerir facilitadores para toda ferramenta
- helpers.h

O Código

O sistema é iniciado através da classe main que encontra-se no arquivo main.cpp, que é o nosso compilado padrão;

```
10  int main()
11  {
12      getMenu("home","0");
13      return 0;
14  }
15
```

Uma vez que o método main é iniciado, este efetuará uma chamada ao método getMenu, que encontra-se no arquivo MenuBO. Esta chamada envia como atributo dois parâmetros, sendo eles menu == home e opc == 0;

O método getMenu(“home”,”0”) retorna o menu Home a tela, e possibilita a escolha de duas opções, sendo elas 1 para menu aviao, e x para sair;
A leitura do método é feita através da variável opc, que é enviada através de return ao método getMenu;

```
15  string getMenuHome(){
16
17          cout << clear;
18          cout << "|--| Aviao[1] | ----------------------------------------| Sair[x] |--|" << endl;
19          cout << "|                                                                    |" << endl;
20          cout << "|                                                                    |" << endl;
21          cout << "|                                                                    |" << endl;
22          cout << "|                                                                    |" << endl;
23          cout << "|                                                                    |" << endl;
24          cout << "|                                                                    |" << endl;
25          cout << "|                                                                    |" << endl;
26          cout << "|                                                                    |" << endl;
27          cout << "|                                                                    |" << endl;
28          cout << "|                                                                    |" << endl;
29          cout << "|                                                                    |" << endl;
30          cout << "|                                                                    |" << endl;
31          cout << "|                                                                    |" << endl;
32          cout << "|                                                                    |" << endl;
33          cout << "|                                                                    |" << endl;
34          cout << "|                                                                    |" << endl;
35          cout << "|                                                                    |" << endl;
36          cout << "|                                                                    |" << endl;
37          cout << "|--------------------------------------------------------------------|" << endl;
38          cout << "Opc.: ";
39
40      string opc;
41      cin >> opc;
42      return opc;
43  }
```

Observe que na linha 49 é feita uma validação para verificar que o resultado da tela getMenuHome retorna 1, caso sim o método getMenu é estanciado por ele mesmo, no entanto desta vez com o atributo menu definido como avião, e não mais como home:

```
45  void getMenu(string menu, string opc){
46
47      if(menu == "home"){
48
49          if(getMenuHome() == "1"){
50              getMenu("aviao", "0");
51          }
52
53      } else if (menu == "aviao") {
```

Uma vez definindo o atributo menu como aviao, o parâmetro opc será utilizado dentro do mesmo, como mostra a imagem abaixo:

```
53      } else if (menu == "aviao") {
54
55          opc = (opc=="0")?getMenuAviao("0"):opc;
56
57          if(opc == "x"){
58              getMenu("home","0");
59          } else if(opc == "1" || opc == "n" || opc == "N"){
60              getAviaoCadastrarView(0);
61              getMenu("aviao","0");
62          } else if(opc == "2"){
63              getMenu("aviao",getAviaoListarView());
64          } else if (opc == "d" || opc == "D") {
65              getDeletarAviaoView();
66              getMenu("aviao",getAviaoListarView());
67          } else if (opc == "e" || opc == "E") {
68              getEditAviaoView();
69              getMenu("aviao",getAviaoListarView());
70          }
71
72      }
```

A variável opc é definida conforme valor do atributo enviado ao método getMenu, desta forma, o atributo opc passa a ter um comportamento orgânico; Uma vez que o valor de opc é zero, a tela getMenuAviao será estanciada, e seu retorno será atribuído a opc, caso contrário, o valor utilizado será o informado a getMenu.

Observe a linha 63, nesta linha o atributo opc do metodo getMenu será o valor informado na tela getAviaoListarView;

Abaixo temos o código de returno do método getAviaoListarView, observe a linha 94

```
86  string getAviaoListarView(){
87      cout << clear;
88      cout << "|---Tela Avioes-----| Edit[e] |--| Delet[d] |--| New[n] |--| voltar[0] |--|" << endl;
89      getAviao();
90      cout << "|-----------------------------------------------------------------Qtd: " << lenght(BaseDAO.avioes) << "-|" << endl;
91      cout << "Opc.: ";
92      string opc;
93      cin >> opc;
94      return opc;
95  }
```

Na linha 89, temos o método getAviao(), que retorna uma lista de avioes cadastrados. Este método é composto no arquivo AvioesBO da seguinte forma:

```
34  int getAviao(){
35      for(int i = 0; i< 9999; i++){
36          if(BaseDAO.avioes[i].id != 0){
37              cout << "| " << BaseDAO.avioes[i].id << " | " << BaseDAO.avioes[i].modelo << " | "
38          }
39      }
40  }
```

Após lista aviões, o código volta para linha 90 e segue as impressões de tela até a linha 93, onde alimenta a variável opc que é retornada ao getMenu na linha 63, onde atribui seu return ao método getMenu(“aviao”, getAviaoListarView()), como mostra o código abaixo:

```
62          } else if(opc == "2"){
63              getMenu("aviao",getAviaoListarView());
```

Este processo se repete em todo menu, variando seu comportamento conforme respostas obtidas na tela getAviaoListarView():

```
57          if(opc == "x"){
58              getMenu("home","0");
59          } else if(opc == "1" || opc == "n" || opc == "N"){
60              getAviaoCadastrarView(0);
61              getMenu("aviao","0");
62          } else if(opc == "2"){
63              getMenu("aviao",getAviaoListarView());
64          } else if (opc == "d" || opc == "D") {
65              getDeletarAviaoView();
66              getMenu("aviao",getAviaoListarView());
67          } else if (opc == "e" || opc == "E") {
68              getEditAviaoView();
69              getMenu("aviao",getAviaoListarView());
70          }
```

O header AvioesBO é fundamental em todas as telas de avião, uma vez que contempla os métodos updateAviao, setAviao, deleteAviao e getAviao:

```
14  int updateAviao(aviaoBean novoAviao){
15      BaseDAO.avioes[(novoAviao.id-1)] = novoAviao;
16      return 1;
17  }
18
19  int setAviao(aviaoBean novoAviao){
20      for(int i = 0; i< 9999; i++){
21          if(BaseDAO.avioes[i].id == 0){
22              BaseDAO.avioes[i] = novoAviao;
23              break;
24          }
25      }
26      return 1;
27  }
28
29  int deleteAviao(int id){
30      BaseDAO.avioes[id].id = 0;
31      return 1;
32  }
33
34  int getAviao(){
35      for(int i = 0; i< 9999; i++){
36          if(BaseDAO.avioes[i].id != 0){
37              cout << "| " << BaseDAO.avioes[i].id << " | " << BaseDAO.avioes[i].modelo << " | " << BaseDAO.avio
38          }
39      }
40  }
```

Estes métodos, são responsáveis por administrar a arrayList de aviaoBean que é colecionada em avioes, dentro da struct BaseDAO, com 9999 posições. Onde aviaoBean é uma struct composta da seguinte forma:

```
7  struct aviaoBean {
8      int id = 0;
9      string modelo;
10     string fabricante;
11     string passageiros;
12     string comprimento;
13     string motor;
14     string altura;
15     string velocidade;
16     string altitude;
17 };
```

```
9  struct data{
10     aviaoBean avioes[9999];
11 } BaseDAO;
```

Acesse o código fonte da aplicação no link:
https://github.com/miamarti/ATPS-cPlusPlus-2s2013

Relatório 1 – Estrutura de Dados

Alocação de memória:

Alocação de memória, consiste no processo de solicitar/utilizar memória durante o processo de execução do programa A alocação de memória pode ser dividida em dois grupos principais:

Alocação Estática: os dados tem um tamanho fixo e estão organizados seqüencialmente na memória do computador. Um exemplo típico de alocação estática são as variáveis globais e arrays;

Alocação Dinâmica: os dados não precisam ter um tamanho fixo, pois podemos definir para cada dado quanto de memória que desejamos usar. Sendo assim vamos alocar espaços de memória (blocos) que não precisam estar necessariamente organizados de maneira seqüencial, podendo estar distribuídos de forma dispersa (não ordenada) na memória do computador. Na alocação dinâmica, vamos pedir para alocar/desalocar blocos de memória, de acordo com a nossa necessidade, reservando ou liberando blocos de memória durante a execução de um programa. Para poder "achar" os blocos que estão dispersos ou espalhados na memória usamos as variáveis do tipo Ponteiro (indicadores de endereços de memória).

O espaço alocado dinamicamente permanece reservado até que explicitamente seja liberado pelo programa. Por isso, podemos alocar dinamicamente um espaço de memória numa função e acessá-lo em outra.
A partir do momento que liberarmos o espaço, ele estará disponibilizado para outros usos e não podemos mais acessá-lo. Se o programa não liberar um espaço alocado, este será automaticamente liberado quando a execução do programa terminar.

Ponteiros em C:

O ponteiro nada mais é do que uma variável que guarda o endereço de uma outra variável.
Ponteiros são uma abstração da capacidade de endereçamento fornecidas pelas arquiteturas modernas. Em termos simples, um endereço de memória, ou índice numérico, é definido para cada unidade de memória no sistema, no qual a unidade é tipicamente um byte, o que em termos práticos transforma toda a memória em um grande vetor. Logo, a partir de um endereço, é possível obter do sistema o valor armazenado na unidade de memória de tal endereço. O ponteiro é um tipo de dado que armazena um endereço.

Estrutura de Dados em C:

Os dados manipulados por um algoritmo podem possuir natureza distinta, isto é, podem ser números, letras, frases etc. Dependendo da natureza de um dado, algumas operações podem ou não fazer sentido quando aplicadas a eles. Por exemplo, não faz sentido falar em somar duas letras - algumas linguagens de programação permitem que ocorra a soma dos valores ASCII correspondentes de cada letra.

Dados Homogêneos
Uma estrutura de dados, que utiliza somente um tipo de dado, em sua deinição é conhecida como dados homogêneos. Variáveis compostas homogêneas correspondem a posições de memória, identificadas por um mesmo nome, individualizado por índices e cujo conteúdo é composto do mesmo tipo. Sendo os vetores
(também conhecidos como estruturas de dados unidimensionais) e as matrizes (estruturas de dados bidimensionais) os representantes dos dados homogêneos.

Vetor
O vetor é uma estrutura de dados linear que necessita de somente um índice para que seus elementos sejam endereçados. É utilizado para armazenar uma lista de valores do mesmo tipo, ou seja, o tipo vetor permite armazenar mais de um valor em uma mesma variável.

Matriz
Uma matriz é um arranjo bidimensional ou multidimensional de alocação estática e seqüencial. A matriz é uma estrutura de dados que necessita de um índice para referenciar a linha e outro para referenciar a coluna para que seus elementos sejam endereçados. Da mesma forma que um vetor, uma matriz é deinida com um tamanho ixo, todos os elementos são do mesmo tipo, cada célula contém somente um valor e os tamanhos dos valores são os mesmos.

Vetores Unidimensionais
Vetores, também conhecidos como arrays, são variáveis que servem para guardar vários valores do mesmo tipo de forma uniforme na memória. Por exemplo, se tivemos que criar 20 variáveis do mesmo tipo que querem dizer a mesma coisa, nós não criaríamos -> int x1, x2, x3, x4, x5, ... ao invés disso, criaríamos apenas uma variável de vetor para guardar todos os 20 números de uma vez. Exato, simples desse jeito.

Como um vetor pode guardar vários valores temos que definir quantos valores ele deve guardar para que seja reservado o espaço necessário em memória. Então, definimos a declaração de um vetor da seguinte maneira:

Primeiro o tipo de dado: int, float, double, ...
Segundo o nome da variável: usando as mesmas convenções de uma variável comum. (array, vetor, variavelDeNumeros, ...)
E por fim, o tamanho necessário do vetor escrito entre colchetes: [5], [10], [3]...

```
34  int main (void){
35      int i, vetor [5];
36      for (i=0; i<=4; i++){
37          cout <<"Digite o "<<i+1<<"o. numero: ";
38          cin >> vetor[i];
39          cin.ignore();
40          system ("cls");
41      }
42      for (i=0; i<=4; i++){
43          cout<<"Posicao "<<i<<" do vetor (vetor ["<<i<<"]) e "<<vetor[i]<<endl;
44      }
45      system ("pause");
46      return EXIT_SUCCESS;
47  }
```

No código acima criamos um vetor de tamanho 5 e usamos uma iteração (repetição / loop) para preencher todos os espaços do vetor e então outra iteração para mostrar todos os valores guardados. Para mudarmos as posições do vetor usamos uma variável chamada i.