Componente 2

Estrutura de Dados e Programação Orientada a objeto

Nome: Gabriel Trita Ferreira da Cunha RA: 345024
Nome: Diego Lima De Freitas RA: 317632
Nome: Rubens Onari Júnior RA: 317373

Turma: Bacharelado de ciências da computação
UFSCAR – Sorocaba.
BCC 09 & BCC 08

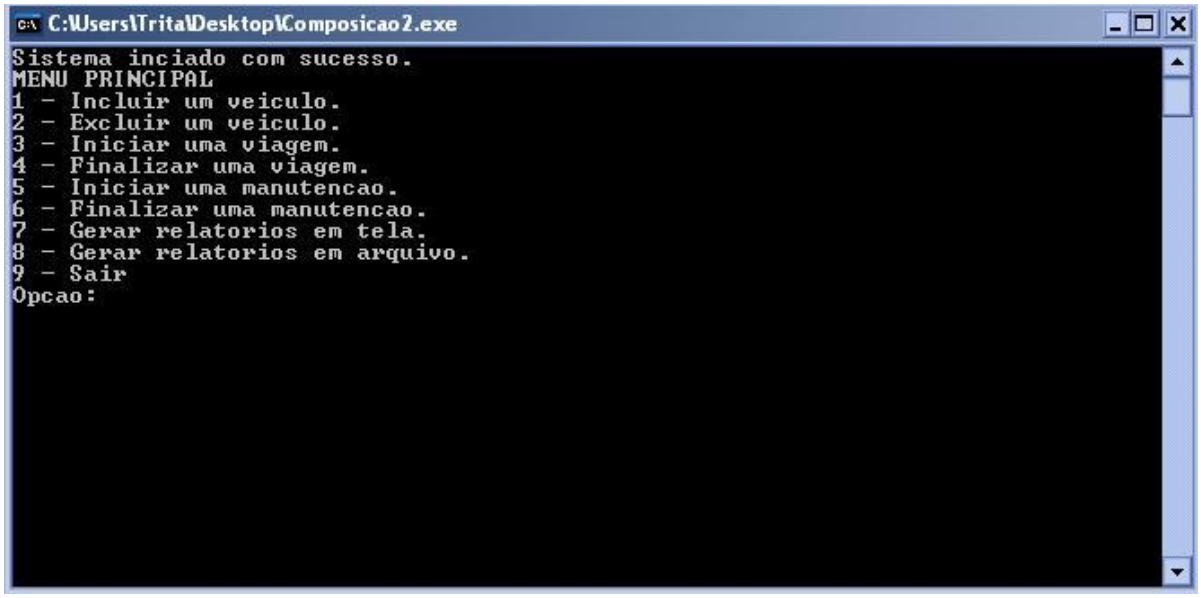
Data de entrega: 10/12/2009

Descrição do programa:

O programa realizado é para controlar um sistema de transportes de veículos, tanto de carga como de passageiros, ele controla as viagens desses veículos, verificando a origem, o destino, a data de inicio, data de término da viagem, o quilometro inicial e final do veiculo utilizado, o nome do motorista/piloto e o número de pedágios existentes na viagem. Controla também a manutenção, verificando a data de inicio e a data de termino da manutenção, veiculo utilizado e a descrição dessa manutenção. Para cadastrar um veiculo precisamos saber inicialmente o tipo dele pode ser um carro, uma van, um ônibus, um caminhão e um avião. Para cadastrar um carro e uma van precisamos saber o combustível usado, o valor do pedágio para eles, o preço do litro do combustível utilizado, o consumo desse combustível por quilometro rodado e o número de passageiros que eles contem. Para cadastrar um ônibus precisamos saber o pedágio pago por ele, o preço do litro de diesel, o preço da taxa rodoviária, o consumo de combustível por quilometro rodado e o numero de passageiros que ele possui. Para cadastrar um caminhão precisa informar o valor do pedágio, o preço do litro de diesel, o consumo de diesel por quilometro rodado, o numero de eixos que ele possui, à capacidade máxima de carga desse caminhão em quilos. Para cadastrar um avião precisamos informar o preço da taxa rodoviária, o custo de combustível do quilometro voado, o numero de funcionários, o salário desses funcionários e a capacidade máxima desse avião. Para fazer a inserção é utilizada uma arvore binária de busca e existem listas para as viagens, manutenções e veículos disponíveis.

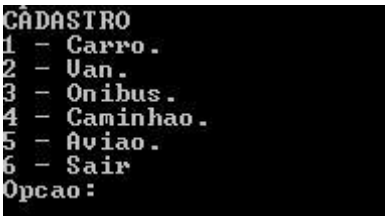
Forma de utilização do programa:

Para utilizar o programa é inicializado um *menu*, onde podemos escolher 8 opções:



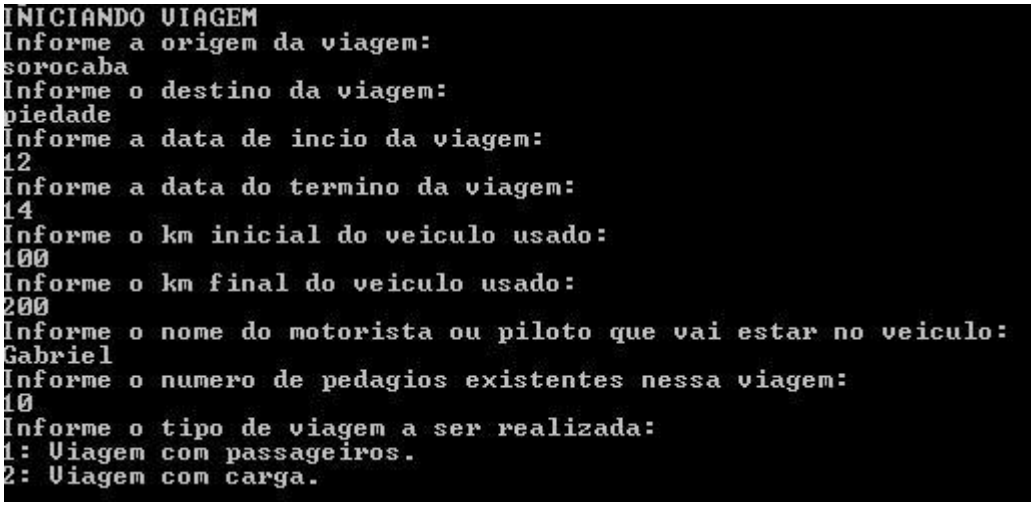
1 – Inclui um novo veiculo no sistema.
2 – Exclui um veiculo já cadastrado do sistema.
3 – Inicia uma viagem utilizando algum veiculo da empresa.
4 – Finaliza uma viagem de algum veiculo da empresa.
5 – Inicia a manutenção de um veiculo já cadastrado da empresa.
6 – Finaliza a manutenção de um veiculo que estava nesse estado.
7 – São gerados vários relatórios na tela.
8 – São gerados vários relatórios em disco (em arquivo).
9 – Finalizar programa.

Caso o usuário queira incluir um veiculo ele devera escolher dentre 6 opções como podemos ver na figura:



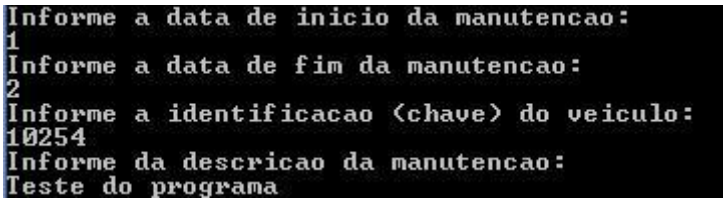
Logo após a escolha desejada é perguntado uma serie de perguntas relacionadas ao tipo de veiculo que o usuário escolheu.

Clicando em iniciar viagem encontramos uma serie de perguntas relacionadas à viagem e logo depois dela é perguntado que tipo de viagem que o usuário deseja realizar, como podemos ver na figura abaixo:



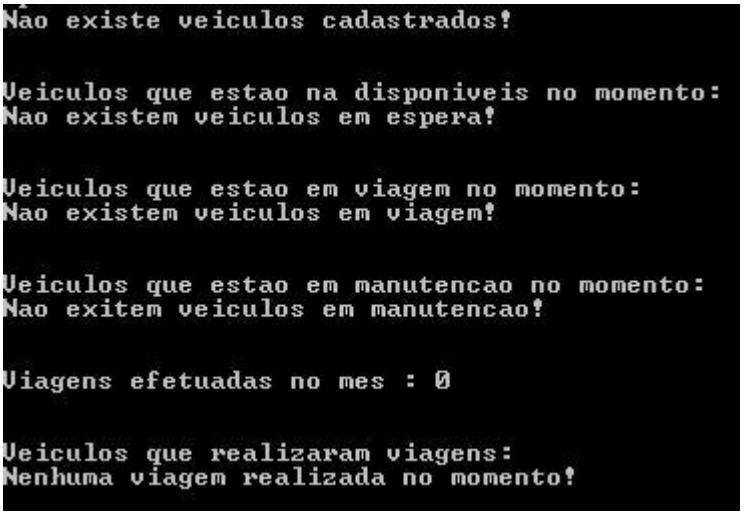
Caso ele queria uma viagem com passageiros, o programa ira selecionar o melhor veiculo para ele dependendo do numero de passageiros que irá à viagem, o mesmo ocorre com o de carga, irá selecionar o melhor veiculo dependendo do numero da carga que ele quer transportar. Logo após de colocar todas as informações necessárias o sistema irá informa se conseguiu ou não realizar a viagem.
Se o usuário quiser finalizar uma viagem ele encontrará a seguinte interface:

Ele terá que informar a chave correspondente ao veiculo que ele quer excluir, com isso pode surgir duas mensagens, uma que ele conseguiu remover ou uma que ele não conseguiu remover.
Caso ele queira realizar uma manutenção, ele irá selecionar o numero cinco que terá a seguinte interface:



Colocando todos os requisitos necessários poderão aparecer duas mensagens, uma que foi realizada com sucesso a manutenção e outra que não foi realizada com sucesso.
Para finalizar uma manutenção o usuário deve digitar seis, que terá a seguinte interface:

Ele terá que informar a chave do veiculo em manutenção.
Para exibir relatórios, basta ele digitar sete (se ele quiser que seja mostrado na tela) ou oito (caso ele queira que seja mostrado em disco).



No exemplo acima o relatório mostra que não tem nenhum veiculo cadastrado, mais se o usuário realizar alguma ação com o programa, tudo será mostrado no relatório.
O relatório em disco é gerado criando seis arquivos, cada um com um relatório diferente, para ficar mais simplificada e mais fácil a visualização.

Descrição dos algoritmos utilizados na resolução do problema proposto

Analisando o programa podemos verificar que temos três grandes problemas, que são:

- Cadastro de veículos
- Viagem de veículos
- Manutenção de veículos

Para solucionar o problema do cadastro de veículos, colocamos no "main.cpp" o menu de cadastro, já foi visto anteriormente, e quando selecionamos um determinado veiculo é perguntado todos os atributos dele.
É criado então um ponteiro para o determinado veiculo:
Exemplo:

```
Carro *novoCarro;
```

Com isso, é realizado um new , que inicializa o construtor do veiculo.

Exemplo:

novoCarro = new Carro(combustivel, pedagio, precoCombustivel, consumoCombustivelKmRodado, numPassageiros);

Inicializando _ o com todas as respostas das perguntas feitas na sua criação. Logo em seguida é chamado um método presente na classe "Sistema.h" que se chama : void cadastrarNovoVeiculo(); ele faz a verificação se é um veiculo de carga ou um veiculo de passageiro, logo em seguida, inseri esse novo veiculo na arvore binária pertencente à classe Sistema.h e na lista de veículos disponíveis. Agora me surgi outro problema, remoção de um veiculo, porem , esse é mais fácil de ser resolvido, fizemos um método chamado: bool descartarVeiculo(); que faz a remoção de um veiculo primeiramente na lista disponível , depois na lista viajando e depois na lista manutenção, isso é claro se ele estiver em alguma lista dessas, logo após ele remove a lista da arvore.
Mais antes de comentar os métodos dos principais problemas iremos comentar sobre algo que nos falamos, porem, não explicamos. Estamos falando sobre a árvore binária e as listas. Não explicamos qual é o tipo delas, vamos inicialmente mostrar como declaramos cada uma:

Exemplo:

```
ABB<int,  Veiculo*> arvore;
Std::list<ABB<int, Veiculo*>::No*> disponivel;
Std::list<ABB<int, Veiculo*>::No*> viajando;
Std::list<ABB<int, Veiculo*>::No*> manutencao;
```

Como podemos ver a nossa arvore binária de busca possui dois tipos, um é chamado de TChave, e o outro é chamado de TDado, o TChave sempre será um inteiro, o TDado sempre será um ponteiro para a classe Veiculo, utilizamos esse ponteiro pois a classe Veiculo é uma classe abstrata, portanto, só aceita ponteiros ou referencias. Agora vamos explicar as listas, toda lista da *stl* tem um tipo de dado, o tipo da nossa é um ponteiro para o No da arvore que possui os tipos de TChave um inteiro, e TDado um Veiculo*. Mais por que um ponteiro para o No da arvore? Porque fica mais fácil de manipulá-lo, alem de não precisar criar dois ou mais objetos retidos. Mais ai você nos perguntaria: Não seria mais fácil fazer um ponteiro somente para a classe Veiculo*? Nossa resposta é que sim, porem, não teríamos acesso a chave pertencente ao No, e achamos que seria arriscado colocar um dado tão importante desse dentro de uma classe,

por exemplo poderia guardar as chaves de todos os carros dentro da classe Carro, porem achamos que ficaria muito inseguro, portanto , adotamos esse método, e ainda iria ficar repetindo uma informação já contida no No, porque teríamos que acrescentar em cada veiculo(carro, van, ônibus, avião e caminhão) um novo atributo, que seria o mesmo que o da chave já contida no No.

Voltando aos principais problemas, iremos comentar sobre o problema das viagens, inicialmente achamos que seria muito fácil e rápido realizar os métodos dessa classe, porem ela nos mostrou que ia ser um desafio muito complicado. Precisávamos de um método para iniciar uma viagem , para isso criamos um método chamado: bool iniciaViagem(), presente na classe "Sistema.h" e "Veiculo.h". Para iniciar os métodos criamos no "main.cpp" uma interface que pergunta todos os atributos que uma viagem inicialmente possui, como já foi visto anteriormente, e perguntamos também se a viagem é uma de passageiros ou de carga. É preciso fazer está pergunta pois é bom diferenciar bem o que seria um veiculo de passageiros(carro, van e ônibus) e um veiculo de carga (caminhão e avião), logo após as perguntas, é chamado o método de iniciarViagem() na classe "Sistema.h", ela verifica se a lista disponível existe algum elemento, se existir ela verifica qual o tipo de viagem que o usuário quer, se é de passageiro (1) ou de carga (2). Vamos fazer o exemplo somente se o veiculo for um de passageiro, mais ele pode ser aplicado quase igualmente , somente uma pergunta muda no de carga e a lista utilizada, sabendo que é um veiculo de passageiros, o método inicia um "for" que corre toda a lista disponível, a procura de algum veiculo, logo após é feito o downcast dos veículos para saber qual será a derivada que será utilizada, logo após de verificar o veiculo utilizado, se inicia outro "for" rodando a lista de números de passageiros(std::list<int> passageiros) , antes vamos explicar está lista e uma lista semelhante, a std::list<float> carga, essas listas foram criadas para armazenar o numero de passageiro e o numero de carga de todos os veículos, mais as senhoras nos perguntam: Mais por que dessas listas? A resposta é bem simples e será melhor entendia no decorrer do método, elas são utilizadas para saber qual é o menor valor de números de passageiros máximos existentes ou o menor numero de carga máxima existente, a inserção nessas listas ocorro no momento em que os veículos são criados, e a ordenação delas é feita sempre que o método da classe "Sistema.h" void iniciarViagem() é chamado. Continuando a explicar o método, é iniciado um "for" na lista de passageiros, em seguida é comparado o menor numero máximo de passageiros com o numero de passageiros que a pessoa quer colocar no veiculo, o "for" vai verificando até o final, até que encontre um numero menor mais que seja maior que o numero de passageiros que o usuário

deseja. Logo em seguida é verificado se esse numero é o mesmo do veiculo que esta sendo analisado, se for, então ele entra no “IF” , ele inseri na lista de todas as viagens (lista necessária para a escrita em relatório) , executa o método da classe veiculo void iniciaViagem(), que só faz a inserção na lista viajando e a retirada na lista disponível, logo em seguida é “setado” todos os atributos da classe Viagem. Isso tudo ocorre de maneira parecida para os veículos de carga.

Agora falaremos da finalização de uma viagem, ela esta presente na classe “Sistema.h” e na classe “Veiculo.h” com o nome de void finalizaViagem().
Ela percorre a lista de viagens na classe “Sistema.h” e realiza o método de finalizarViagem() na classe Veiculo, ela consegue localizar a viagem correta pelo fato de analisar a chave, com isso é removido da lista de viajando, viagens e inserido na lista de disponível.

Agora vamos comentar o problema da manutenção, esse de fato foi o problema mais fácil de ser solucionado, ele utiliza 2 métodos, um presente na classe “Sistema.h” e outro presente na classe “Veiculo.h”, mais é basicamente é inserção na lista manutenção, para inserir uma nova manutenção e remoção na lista manutenção pra realizar a finalização da mesma.

Descrição das condições de contorno

Entradas que ocorrem um laço infinito(loop) caso ocorra a digitação de um caractere ao invés de um numero:

- No momento do cadastro de algum veiculo:
  - Informe um valor de pedágio para este (veículo selecionado):
  - Informe o preço do litro de (combustível informado) para o (veículo selecionado):
  - Informe o consumo de combustível por Km rodado pra este (veículo selecionado):
  - Informa o número de passageiros que este carro contém:
  - Informe a chave do veículo que você queria dar baixa:
  - Informe a data de inicio da viagem:
  - Informe a data de término da viagem:
  - Informe o km inicial do veiculo usado:
  - Informe o número de pedágios existentes nessa viagem
  - Informe o número de passageiros:

- Informe a distancia percorrida pelo veículo na viagem:
- Informe o número de carga existente no veículo em kg:
- Informe a data de inicio da manutenção:
- Informe a data do inicio do fim da manutenção:
- Informe a identificação (chave) do veiculo:

Esses itens só aceitam inteiros OU reais, nota-se que todas as funções desempenhadas pelo programa funcionam de forma correta, porém é preciso bom senso do usuário de modo a informar os dados de entrada corretamente.
Um controle maior de exceções poderia aperfeiçoar o programa, porém tendo em vista que as funcionalidades do sistema não acarretaram erros, basta se adaptar as entradas e tudo acontecerá normalmente.

Descrição dos testes realizados

Menu Pricipal
1-Incluir veiculo
2-Excluir veiculo
3-Iniciar uma viagem
4-Finalizar uma viagem
5-Iniciar uma manutenção
6-Finalizar uma manutenção
7-Gerar relatórios em tela
8-Gerar relatórios em arquivo
9-Sair

Incluir veiculo (Cadastro)
1-Carro
2-Van
3-Onibus
4-Caminhão
5-Avião
6-Sair

Todas as opções do menu principal e do menu de cadastro foram testadas, sendo que todas corresponderam da forma esperada, executando suas devidas funções.

OBS: Se algo diferente de um número inteiro for digitado, o programa entra em loop infinito.

Após selecionar o veículo, uma seqüência de entradas é solicitada:

1-Informe a chave do veículo, somente números.
Resultado Obtido: Se entrar com um número inteiro, funciona corretamente, se entrar com um número real, o programa pula a próxima solicitação de entrada. Entra em loop infinito se digitar um caractere qualquer.

Há um controle da chave, pois caso uma chave que já está cadastrada seja informada, acusa um erro e solicita novamente outra chave.

2-Informe o tipo de combustível que esse carro contém:
Resultado Obtido: A entrada com caracteres funciona corretamente, o programa também identifica um número inteiro ou real como entrada válida.

3-Informe um valor de pedágio para este carro:
Resultado obtido: Entrar com números inteiros ou reais, funciona da forma correta. Entra em loop infinito se digitar um caractere qualquer.

4-Informe o preço do litro de (combustível informado) para o (veículo selecionado):
Resultado Obtido: Entrar com números inteiros ou reais, funciona da forma correta. Entra em loop infinito se digitar um caractere qualquer.

5-Informe o consumo de combustível por Km rodado pra este (veículo selecionado):
Resultado Obtido: Entrar com números inteiros ou reais, funciona da forma correta. Entra em loop infinito se digitar um caractere qualquer.

6-Informa o número de passageiros que este carro contém:
Resultado Obtido: Entrada com números inteiros é a única válida, caso entra com outro tipo de entrada, o programa entra em loop infinito.

As solicitações mostradas acima, são utilizadas para veículos que levam passageiros, veículos de carga, ou até mesmo caminhões possuem uma seqüência um pouco diferente de entradas, mas o padrão em que se deve entrar com os dados deve ser mantido, informando caracteres só quando necessário ou solicitado.

Após entrar com todos os dados solicitados, o programa emite uma mensagem de que o veículo foi cadastrado, e retorna ao menu inicial.

Ao selecionar a opção 2 do menu principal (Excluir veículo), a seguinte mensagem é apresentada:

1-Informe a chave do veículo que você queria dar baixa:
Resultado Obtido: Se informar um número inteiro, a chave do veículo deve estar cadastrada para que a mensagem de veículo removido com sucesso seja apresentada, caso contrário, acusará que o veículo não pode

ser removido. Se entrar com um número real ou caractere, entra em loop infinito.

Após informar a chave do veículo retorna ao menu inicial.

Ao selecionar a opção 3 do menu principal (Iniciar uma viagem) as seguintes mensagens são apresentadas:

1-Informe a origem da viagem:
Resultado Obtido: Aceita a entrada de qualquer tipo de dado (string, caractere, inteiro, real), funcionando corretamente.

2-Informe o destino da viagem:
Resultado Obtido: Aceita a entrada de qualquer tipo de dado (string, caractere, inteiro, real), funcionando corretamente.

3-Informe a data de inicio da viagem:
Resultado obtido: Entra em loop infinito se a entrada de dados ocorrer nos seguintes formatos:
(10/01/1989), (10/01), caracteres ou valores reais. Só aceita valores inteiros como 10, por exemplo.

4-Informe a data de término da viagem:
Resultado obtido: Entra em loop infinito se a entrada de dados ocorrer nos seguintes formatos:
(10/01/1989), (10/01), caracteres ou valores reais. Só aceita valores inteiros como 10, por exemplo.

5-Informe o km inicial do veiculo usado:
Resultado obtido: Aceita valores reais e inteiros como entrada, de modo a armazená-los corretamente, mas entra em loop infinito se entrar com caracteres.

6-Informe o nome do motorista ou piloto que vai estar no veículo:
Resultado obtido: Pode-se informar o nome correto com espaços que será armazenado, mas se informar um número qualquer por exemplo, também armazena o dado.

7-Informe o número de pedágios existentes nessa viagem:
Resultado obtido: Entrada com valores que não sejam inteiros acarretam em erro, gerando um loop infinito do programa.

8-Informe o tipo de viagem a ser realizada:

8.1-Viagem com passageiros:

8.1.1-Informe o número de passageiros:
Resultado Obtido: Entrada com valores que não sejam inteiros acarretam em erro, gerando um loop infinito do programa.

8.1.2-Informe a distancia percorrida pelo veículo na viagem:
Resultado Obtido: Aceita valores reais e inteiros, mas caractere entra em loop infinito.

Ao final, caso nenhuma entrada tenha sido fornecida de forma errada, acusa que a viagem foi iniciada com sucesso. Mas se nenhum veículo de passageiros foi cadastrado, acusa que a viagem não pode ser iniciada.

8.2-Viagem com carga:

8.2.1-Informe o número de carga existente no veículo em kg:
Resultado Obtido: Lê números reais e inteiros, mas não aceita caracteres.

Ao final, independente das entradas, sempre acusa que a viagem não pode ser iniciada, caso um veículo de carga não tenha sido cadastrado. Para que uma viagem seja cadastrada, é necessário que um veículo de carga seja cadastrado.

Ao escolher a opção 4 do menu principal (Finalizar uma viagem), a solicitação da chave do veículo será apresentada, a viagem será removida, caso exista uma viagem cadastrada com o número da chave informado, senão, não irá finalizar a viagem.

Escolhendo a opção 5 do menu principal (Iniciar manutenção), as seguintes solicitações serão apresentadas:

1-Informe a data de inicio da manutenção:
Resultado obtido: Entra em loop infinito se a entrada de dados ocorrer nos seguintes formatos:
(10/01/1989), (10/01), caracteres ou valores reais. Só aceita valores inteiros como 10, por exemplo.

2-Informe a data do inicio do fim da manutenção:
Resultado obtido: Entra em loop infinito se a entrada de dados ocorrer nos seguintes formatos:
(10/01/1989), (10/01), caracteres ou valores reais. Só aceita valores inteiros como 10, por exemplo.

3-Informe a identificação (chave) do veiculo:
Resultado Obtido: Um número inteiro deve ser informado, caso o contrário, ocorrerá um erro no programa.

4-Informe a descrição da manutenção:
Resultado Obtido: Um número inteiro deve ser informado aqui, mesmo que pareça que um texto deve ser informado, erros irão acontecer caso insira um tipo de dado diferente.

Ao Final, o número de chave e o número da manutenção são apresentados, uma mensagem de que a manutenção foi iniciada com sucesso, caso exista um veículo cadastrado com o número de chave informado, também será apresentada. Caso contrário, uma mensagem de que a manutenção não foi iniciada é emitida.

Ao selecionar a opção 6 do menu principal (Finalizar manutenção) a solicitação da chave do veículo será apresentada, a manutenção será removida, caso exista uma manutenção cadastrada com o número da chave informado, senão, não irá finalizar a manutenção.

Selecionando a opção 7 do menu principal (Gerar relatórios em tela), todas as informações cadastradas até então são apresentadas de forma correta, e caso não possua dados a serem apresentadas, mensagens acusando a falta dos mesmos são apresentadas.

Selecionando a opção 8 do menu principal (Gerar relatórios em arquivos), todas as informações cadastradas são geradas de forma correta em arquivos .txt, o usuário pode visualizá-los na mesma pasta que contém o projeto com seus respectivos arquivos .h e .cpp.

Selecionando a opção 9 do menu principal (Sair), o programa é finalizado de forma correta.

Dificuldades encontradas

Inicialmente tivemos uma grande dificuldade para entender a parte da herança, tivemos que colocar no papel todas as classes para definir quais os atributos pertenciam a que classe, podemos chegar à seguinte conclusão:
Atributos de todos os veículos:
Carro:

- `ClasseDeExcecao<float> *erro;` (Herda da classe Veiculo)

- ClasseDeExcecao<int> *erroi; (Herda da classe Veiculo)
- int capacidadeMaxPassageiros; (Herda da classe VeiculoPassageiro)
- int numPassageirosTransportadosViagemAtual; (Herda da classe VeiculoPassageiro)
- float pedagio; (Herda da classe VeiculoPassageiro)
- float precoCombustivel; (Herda da classe VeiculoPassageiro)
- float consumoCombustivelKmRodado; (Herda da classe VeiculoPassageiro)
- std::string combustivel; (Herda da classe VeiculoPassageiro)

Van:

- ClasseDeExcecao<float> *erro; (Herda da classe Veiculo)
- ClasseDeExcecao<int> *erroi; (Herda da classe Veiculo)
- int capacidadeMaxPassageiros; (Herda da classe VeiculoPassageiro)
- int numPassageirosTransportadosViagemAtual; (Herda da classe VeiculoPassageiro)
- float pedagio; (Herda da classe VeiculoPassageiro)
- float precoCombustivel; (Herda da classe VeiculoPassageiro)
- float consumoCombustivelKmRodado; (Herda da classe VeiculoPassageiro)
- std::string combustivel; (Herda da classe VeiculoPassageiro)

Ônibus:

- ClasseDeExcecao<float> *erro; (Herda da classe Veiculo)
- ClasseDeExcecao<int> *erroi; (Herda da classe Veiculo)
- int capacidadeMaxPassageiros; (Herda da classe VeiculoPassageiro)
- int numPassageirosTransportadosViagemAtual; (Herda da classe VeiculoPassageiro)
- float pedagio; (Herda da classe VeiculoPassageiro)
- float precoCombustivel; (Herda da classe VeiculoPassageiro)
- float consumoCombustivelKmRodado; (Herda da classe VeiculoPassageiro)
- std::string combustivel = “DIESEL” (DEFAUL , não sera inicializado na hora de cria-lo);
- float taxaRodoviaria; (Atributo de sua classe)

Caminhão:

- ClasseDeExcecao<float> *erro; (Herda da classe Veiculo)
- ClasseDeExcecao<int> *erroi; (Herda da classe Veiculo)

- `float capacidadeMaxCarga; (Herda da classe VeiculoCarga)`
- `float pesoCargaTransportada; (Herda da classe VeiculoCarga)`
- `float diesel; (Atributo de sua classe)`
- `int numEixos; (Atributo de sua classe)`
- `float pedagio; (Atributo de sua classe)`
- `float consumoCombustivelKmRodado; (Atributo de sua classe)`

Avião:

- `ClasseDeExcecao<float> *erro; (Herda da classe Veiculo)`
- `ClasseDeExcecao<int> *erroi; (Herda da classe Veiculo)`
- `float capacidadeMaxCarga; (Herda da classe VeiculoCarga)`
- `float pesoCargaTransportada; (Herda da classe VeiculoCarga)`
- `float taxaAeroporto; (Atributo de sua classe)`
- `float custoKmVoado; (Atributo de sua classe)`
- `int numFuncionarios; (Atributo de sua classe)`
- `float salarioFuncionarios; (Atributo de sua classe)`
- `float numHorasVoadas; (Atributo de sua classe)`

A segunda e ultima dificuldade foi mesmo para realizar a viagem, pois teria que escolher o melhor veiculo pra determinado numero de passageiros ou de carga, e isso foi resolvido com duas lista ordenadas.
Tirando esses dois, acho que tudo ocorreu como o esperado.

Possíveis extensões e melhorias no programa

Poderíamos aumentar o numero de exceções tratadas quando um inteiro ou um real são pedidos e o usuário digita um valor invalido e causa um erro no programa, poderíamos melhorá-lo dessa forma.

Comentários pessoais

O programa não era muito difícil de ser feito, porem era trabalhoso, e com ele podemos frisar bem a parte de herança, pois essa parte sim foi complicada, ela nos ajudou compreender perfeitamente como funciona essa parte tão interessante da programação orientada a objeto.