premissas assumidas

- Linguagem de programação utilizada será a linguagem C++ por ser orientada a objeto.

- Devido ao modelo de entrada de valores a data deverá ser tratada para que seja possível descobrir a qual dia da semana ela pertence.

- Graças ao detalhamento do problema é possível notar 3 modos diferentes de cobranças nos petShops:

1) O petShop possui um valor específico para os finais de semana sendo que, tal valor corresponde a uma variação percentual dos valores cobrados em dias úteis.

2) O petShop possui um valor específico para os finais de semana sendo que, tal valor corresponde a um valor fixo assim como os cobrados em dias úteis.

3) O petShop não possui um valor específico para os finais de semana.

- Existem valores diferentes para animais de grande porte e animais de pequeno porte. Dessa forma, é necessária a separação desses atributos.

- Baseado nas premissas anteriores, faz-se necessária a criação de uma classe contendo métodos polimórficos para que os mesmos possam atender a todo tipo de cobrança presente nos petShops.

- Devido a quantidade de valores a serem tratados a aplicação de um teste funcional pode ajudar a identificar se os valores gerados são verdadeiros.

Decisões de projeto

- A primeira grande decisão do projeto foi a linguagem a ser utilizada. Pelo fato de conhecer a linguagem C++ a mais tempo e conhecer seu potencial e "leveza" do ponto de vista computacional achei que seria uma boa opção.

- Outra decisão que diz respeito ao desenvolvimento foi a escolha do editor de texto Sublime e o sistema operacional baseado em Linux por ser mais rápido e de melhor desempenho que o Windows.

- Uma decisão muito importante a ser tomada foi a de como tratar a data devido seu formato de entrada. Para tratar datas fiz uma pesquisa das funções presentes na biblioteca "time" da linguagem C e percebi que através da "struct tm" era possível fazer cálculos que diriam com precisão qual o dia da semana correspondente. Os membros dessa estrutura a serem utilizados foram:

tm_mday	dias do mês	1-31
tm_mon	messes desde janeiro	0-11
tm_year	anos desde 1900	

- Para realizar os cálculos criei uma função para separar os valores presentes na string da data e excluindo os caracteres "/". após a separação dos números a função localtime() converte um "time_t"(estrutura de tempo) em horário local e com isso, calculasse o tempo a partir dos dados de entrada. Ao final do processo a função mktime() faz o processo inverso voltando com o formato estrutural. Através de uma constante contendo o nome dos dias é possível fazer a "conversão" da estrutura de data padrão para string contendo o nome do dia da semana.

referência detalhada: http://www.cplusplus.com/reference/ctime/tm/

- Outra decisão de projeto tomada foi em relação a estrutura de dados a ser utilizada. A classe petShop deveria atender a todas as possibilidades previstas nas premissas assumidas. Portanto, foi necessária a criação de 4 construtores:

1) Para os petShops que não tem diferença no preço do final de semana portanto, não possuía nenhuma informação além da distância, preço para animais pequenos, preço para animais grande, a quantidade dos mesmos e algumas outras informações padrões como nome. (atributos não utilizados foram zerados).

2) Para os petShops que possuem diferença no preço do final de semana baseado em uma porcentagem o construtor aceita a entrada desse valor percentual para os cálculos.

3) Para os petShops que possuem diferença no preço do final de semana baseado em um valor fixo o construtor recebe além dos valores dos dias úteis o valor correspondente ao final de semana.

- Outras ocorrências de polimorfismo vieram dos métodos "set" que precisam saber a qual dia da semana estamos trabalhando para poder utilizar os preços adequados. Para tanto foi feita uma verificação do nome do dia tratado com o nome dos finais de semana presentes na estrutura weekdays.

- Para saber a melhor opção de petShop a função selectPetShop() conta com o apoio da função ordena() que coloca em ordem crescente dos valores finais o vetor de petShop. Com isso a função apenas precisa verificar se as duas primeiras posições estão empatadas. Se sim, pega o menor valor em distância como critério de desempate.

- No que diz respeito aos testes funcionais foram criados métodos assertivos para verificar a saída de informações a partir de uma entrada previamente programada. Tal teste foi possível de ser realizado através da função assert() que verifica se o retorno da função corresponde com o retorno esperado.