UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA
CIÊNCIA DA COMPUTAÇÃO

LEONARDO NASCIMENTO DOS SANTOS
VINÍCIUS BERGER

DESENVOLVIMENTO DE SOFTWARE SIMPLES PARA LEITURA E ADMINISTRAÇÃO DE CONSUMO DE ENERGIA ELÉTRICA

VITÓRIA
2015

LEONARDO NASCIMENTO DOS SANTOS
VINÍCIUS BERGER

DESENVOLVIMENTO DE SOFTWARE SIMPLES PARA LEITURA E ADMINISTRAÇÃO DE CONSUMO DE ENERGIA ELÉTRICA

Trabalho apresentado à disciplina Estrutura de Dados I, do curso superior em Ciência da Computação da Universidade Federal do Espírito Santo, como requisito para obtenção da avaliação na disciplina.
Professor: Thomas Walter Rauber.

VITÓRIA
2015

1 RESUMO

O script criado lê um arquivo de extensão tipo “*txt”*, que contém as instruções de entrada para a inserção e manipulação dos dados no sistema e gera um arquivo de saída denominado *“protocolo_saida.txt”*, contendo em cada linha o resultado do processamento da instrução correspondente.

2 INTRODUÇÃO

O projeto de um programa engloba, entre outras, a fase de identificação das propriedades dos dados e suas características funcionais. Uma representação adequada dos dados, em vista das funcionalidades que devem ser atendidas, constitui uma etapa fundamental para a obtenção de programas eficientes e confiáveis.

O Tipo Abstrato de Dado (TAD) é uma especificação de um conjunto de dados e operações que podem ser executadas sobre esses dados. Além disso, é uma metodologia de programação que tem como proposta reduzir a informação necessária para a criação/programação de um algoritmo através da abstração das variáveis envolvidas em uma única entidade fechada, com operações próprias à sua natureza. Assim, o TAD procura encapsular de quem usa determinado tipo a forma concreta com que o tipo foi implementado.

Um exemplo prático da aplicação deste conceito no atual projeto é a especificação do tipo que se refere a uma casa. Sem a aplicação da teoria do TAD, uma casa seria representada por variáveis soltas (como id da casa, número da casa, id da rua, nome do consumidor, consumo de energia) que seriam operadas separadamente, sem ligação lógica entre elas. Além disso, seria necessário um conhecimento prévio por parte do programador de que as variáveis mencionadas referem-se à entidade “casa”.

Por outro lado, utilizando a teoria do TAD, um programa pode ser projetado sem que o programador se preocupe com as variáveis isoladas, mas simplesmente com o tipo que as envolve.

Este novo tipo, como um tipo simples qualquer (inteiro ou string, por exemplo), deve ter operadores próprios. Assim, o tipo casa deve possuir operações desejáveis ao programador, como inserir uma nova casa, remover uma casa, medir o consumo de energia, dentre outras.

Dessa forma, visando atender o objetivo do projeto, desenvolveu-se um software baseado na linguagem de alto nível C, com o uso de tipos de dados abstratos, estrutura de dados e alocação dinâmica de memória. Ferramentas como o *Notepad++* e o *Gedit* foram utilizadas para a criação e edição do código, enquanto a ferramenta *Valgrind* foi utilizada para a verificação de vazamento de memória.

3 OBJETIVOS

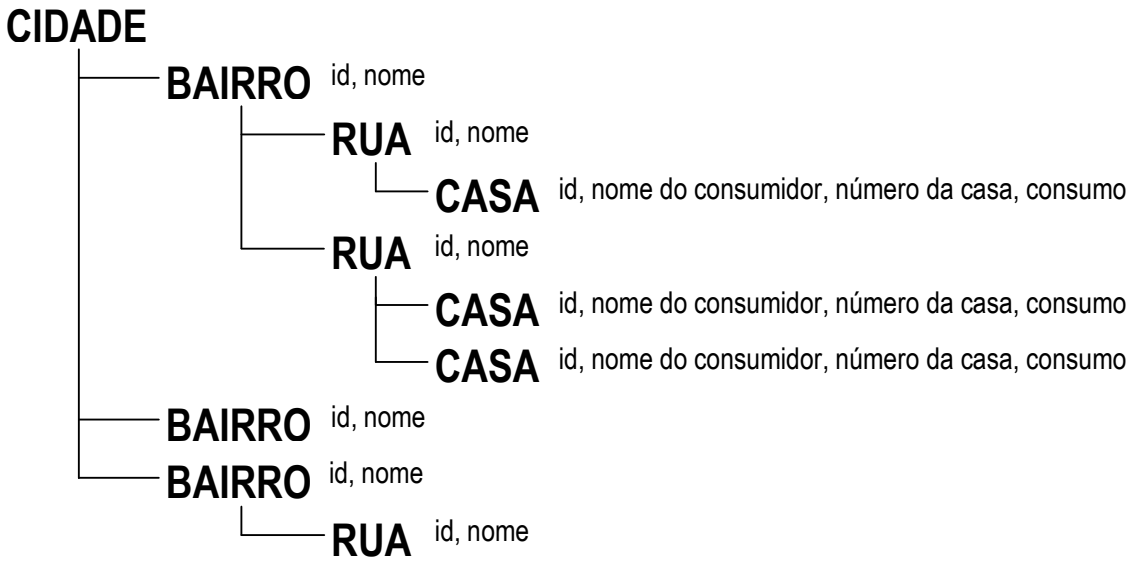
Este projeto tem como objetivo geral a criação de um *software* capaz de receber instruções provenientes de um arquivo de entrada do tipo *"txt"*, realizar o processamento adequado (inserir ou remover uma informação do sistema, por exemplo) e gerar um relatório de saída contendo os resultados referentes ao processamento.

3.1 OBJETIVOS ESPECÍFICOS

- Representação e manipulação de informação estruturada por linguagem de programação de alto nível.
- Desenvolver o conhecimento das estruturas de dados, com o uso de listas encadeadas simples e dos tipos abstratos de dados.

4 METODOLOGIA

Primeiramente foi realizado um diagrama do sistema para facilitar sua visualização e entendimento do problema.



Em seguida, foram implementadas as estruturas referentes a cada unidade, utilizando listas homogêneas simplesmente encadeadas, de acordo com a especificação do projeto.

As funções referentes a cada unidade foram então implementadas, garantindo todas as funcionalidades exigidas na especificação. A ferramenta *Valgrind* foi utilizada a fim de verificar a rigidez na administração da memória alocada dinamicamente pelas funções de inclusão.

Uma lista de erros padronizada foi criada para que cada função retorne um código específico de erro, pois várias funções compartilham dos mesmos erros (impossibilidade de inserir um id negativo, por exemplo).

Na função principal (*main*) foi criada a interface do software e as chamadas às funções de incluir bairro, leitura de arquivo e liberar a memória alocada.

Por fim, a metodologia de programação baseada em tipos abstratos de dados foi implementada para encapsular a implementação das estruturas e operações relacionadas a elas. Além disso, diversos modelos de arquivos-entrada foram desenvolvidos a fim de contemplar todos os possíveis erros do sistema.

As ferramentas utilizadas no desenvolvimento do projeto estão listadas a seguir:

- Linguagem de alto nível C;
- Sistema operacional Linux Mint;
- Compilador gcc (GNU Compiler Collection);
- Depurador Valgrind;
- Bibliotecas padrão stdlib.h, stdio.h e string.h
- Editores Notepad++ e Gedit.

4.1 ESTRUTURA

4.1.1 ORGANIZAÇÃO DA INFRAESTRUTURA

O software foi projetado para realizar tarefas relacionadas às seguintes entidades: Cidade, Bairro, Rua e Casa.

UNIDADE	PARÂMETROS
Cidade	Lista de Bairros*
Bairro	Id, Nome, Lista de Cidades*
Rua	Id, Nome, Lista de casas*
Casa	Id, Número da casa, Consumo, Nome do consumidor

Cada unidade contém um identificador numérico inteiro positivo, além de informações complementares úteis. Como simplificação assume-se que não existam prédios com múltiplas unidades, somente casas com um único consumidor.

A estrutura das unidades é hierárquica. Uma cidade tem vários bairros; um bairro tem várias ruas; uma rua tem várias casas. Como simplificação assume-se que não existam ligações entre casas de diferentes ruas e ruas entre diferentes bairros.

4.1.3 REPRESENTAÇÃO DA INFORMAÇÃO

As unidades cidade, bairro e rua foram organizadas como listas encadeadas, ou seja, a cidade é uma lista de bairros encadeados, um bairro é uma lista de ruas e uma rua é uma lista de casas. Dessa forma, cada unidade do sistema foi implementada como uma estrutura de dados, que contém, basicamente:

- Um ponteiro para o primeiro elemento da lista de unidades (hierarquicamente um nível inferiores) que estão inseridas na atual;
- Informações sobre a unidade (id e nome, por exemplo);
- Um ponteiro para o próximo elemento da mesma unidade;

A cidade foi implementada como ponteiro para bairro, ou seja, não há informações nem ponteiro para a próxima cidade, somente para o primeiro elemento da lista de bairros.

Levando em consideração que os bairros não mudam neste projeto, a inicialização da cidade foi feita por meio de funções de inicialização fixas, que inserem vários bairros na cidade. Ainda assim, se no arquivo de entrada houver uma instrução para inserir bairro, esta será processada e efetivada caso não haja erro.

4.1.2 FUNÇÕES

O sistema provê funcionalidades administrativas e operacionais. Por exemplo: como administração, é possível incluir uma nova rua dentro de um bairro, ou uma nova casa dentro de uma rua de um determinado bairro. Como função operacional é possível, por exemplo, medir o consumo de um consumidor, rua, bairro e da cidade.

A tabela a seguir apresenta todas as funcionalidades que o software disponibiliza ao usuário. Assim, qualquer instrução no arquivo de entrada que se referir a uma dessas funcionalidades será processada e efetivada (exceto em caso de erro).

UNIDADE	FUNÇÃO	PARÂMETROS
ADMINISTRATIVAS		
BAIRRO	Incluir na cidade	Id bairro
RUA	Incluir no bairro	Id bairro, id rua, nome
RUA	Eliminar do bairro	Id bairro, id rua
CASA	Incluir na rua	Id bairro, id rua, id casa, número da casa, consumo, nome do consumidor
CASA	Eliminar da rua	Id bairro, id rua, id casa
OPERACIONAIS		
CASA	Consumir	id bairro, id rua, id casa, Consumo
CASA	Medir consumo	id bairro, id rua, id casa
RUA	Medir consumo	id bairro, id rua
BAIRRO	Medir consumo	id bairro
CIDADE	Medir consumo	

Vale ressaltar que as funções de inserção alocam dinamicamente memória para o armazenamento dos novos elementos, ou seja, as estruturas não são pré-dimensionadas. O número de elementos que podem ser representados é arbitrário.

Uma exigência explícita na especificação do projeto é que a ordem das casas em uma lista seja mantida de forma crescente. Dessa forma, quando uma casa for introduzida no sistema, a ordem da enumeração será preservada. Por exemplo, se existirem as casas com os números 12 e 18, e houver uma solicitação para inserção da casa número 16, esta será introduzida entre as duas existentes.

O sistema é robusto em relação a inconsistências. Cada uma das dez funções mencionadas na tabela anterior retorna um código de erro/sucesso que é utilizado para fornecer informações ao usuário sobre a inconsistência encontrada no

comando. Se o usuário tentar inserir uma rua em um bairro que não existe, por exemplo, o sistema não realizará a inclusão e retornará um erro.

4.1.4 INTERFACE

Toda a interação funciona através de arquivos de entrada e saída. A sintaxe dos comandos do arquivo é a seguinte:

<Unidade> <Ação> <Parâmetros>

Exemplos:
rua incluir 17 3 "Av. Fernando Ferrari"
casa eliminar 17 23 8
casa consumir 17 23 8 255
bairro medir 17
cidade medir

Os elementos da linha de comando são separados por espaço em branco ou caractere de tabulação. Nomes em forma de cadeias de caracteres devem aparecer entre aspas, como no exemplo acima. O protocolo de saída emite uma confirmação ou rejeição, em caso de inconsistência, para cada comando. Por exemplo: o pedido de medir o consumo de uma rua retorna, em caso de comando válido, id do bairro, id da rua e consumo total.

4.2 FUNCIONAMENTO

Inicialmente o programa importa as bibliotecas necessárias para se trabalhar com entradas e saídas de dados (stdio.h), alocação de memória (stdlib.h) e manipulação de cadeia de caracteres (string.h), além do arquivo de protótipo (tad.h) do módulo (tad.c).

Ao iniciar o programa a variável global “Cidade” é inicializada com “NULL” a fim de, assim que for necessário, receber o endereço para o primeiro elemento da lista de bairros. Isso é necessário porque a função que insere bairros irá inserir cada novo bairro no início da lista. Caso houvesse um endereço qualquer (lixo) na variável “Cidade” antes da inserção do primeiro bairro, no momento da inserção este endereço seria interpretado como o próximo elemento da lista, o que faria com que o software não identificasse o final da lista.

Uma interface de “boas vindas” é então apresentada ao usuário, solicitando, logo em seguida, que seja informado o nome do arquivo de entrada (arquivo.txt), que deve estar no mesmo diretório do programa e conter as instruções a serem processadas pelo software. A string digitada (input do usuário), que representa o nome do arquivo, é armazenada em uma variável a fim de ser utilizada posteriormente como parâmetro para a função de ler arquivo.

O software insere automaticamente quatro bairros no sistema, como sugerido na especificação do projeto. Isso não impede que outros bairros sejam inseridos por meio dos comandos dados no arquivo de entrada.

Logo em seguida, a função principal do programa “chama” a função que lê um arquivo, passando como parâmetro a string armazenada anteriormente.

4.2.1 A FUNÇÃO DE LEITURA DE ARQUIVO

A função de leitura de arquivo é responsável pela abertura do arquivo informado pelo usuário, leitura dos comandos, execução das funções correspondentes a cada instrução e geração do protocolo de saída, que contém o resultado do processamento de cada comando.

O primeiro bloco de instruções realiza a abertura de dois arquivos, um para representar o arquivo de entrada e o outro o protocolo de saída, utilizando, para isso, duas variáveis-ponteiros do tipo FILE. Caso haja algum problema na abertura, o sistema imprime na tela uma mensagem de erro e aborta o programa.

Em seguida, uma estrutura de repetição “while” é utilizada para percorrer o arquivo, lendo palavra por palavra até encontrar a constante EOF (End of File), que é definida na biblioteca “stdio.h” e é devolvida à função chamadora quando não existem mais caracteres a serem lidos no arquivo. Quando isso ocorrer, os dois arquivos abertos serão fechados, retornando o comando do programa para a função principal (*main*).

Caso a constante EOF não tenha sido retornada, a função verifica se a string lida é uma unidade válida (cidade, bairro, rua ou casa), retornando um erro no protocolo de saída em caso negativo. Se não houver erro o sistema deixa a unidade armazenada na variável atual e prossegue a leitura da próxima palavra.

De acordo com a sintaxe dos comandos, a próxima palavra refere-se à ação que deve ser realizada. Dessa forma, o sistema verifica se a string lida corresponde a uma ação válida e retorna um erro ou armazena a string e prossegue a leitura dos parâmetros, conforme o caso.

O terceiro argumento da sintaxe dos comandos apresenta os parâmetros que serão utilizados na execução das funções do sistema. A quantidade de parâmetros varia de função para função. Dessa forma, a leitura é interrompida somente quando é encontrado um caractere de quebra de linha, significando que a estrutura do comando já foi completamente lida e armazenada.

Após o término da leitura da instrução, o software executa a função correspondente e “escreve” o resultado do processamento em uma nova linha do protocolo de saída.

Ao terminar de ler e executar todos os comandos do arquivo de entrada, a função fecha os arquivos abertos e devolve o controle do software para a função principal (*main*), a qual irá liberar todos os espaços alocados na memória e encerrar o programa.

4.3 ERROS TRATADOS

O tratamento dos erros acontece por meio do retorno de cada função e possíveis chamadas dentro da função que lê um arquivo. Uma lista com várias mensagens (de erro e de sucesso) foi definida no protótipo (tad.h) do módulo (tad.c). A função de leitura de arquivo é responsável por receber o código de retorno de cada função executada e relacionar ao código de erro definida na lista mencionada.

Por exemplo, se uma determinada função retornar o código 16 (ou -16.00f, para funções float) significa que o id do bairro fornecido no comando é negativo, o que torna o comando inválido e gera uma mensagem de erro (código 16) no protocolo de saída.

A função de leitura de arquivo também pode chamar uma mensagem de erro sem precisar de um código de retorno de função. Por exemplo, se ao tentar ler uma unidade (primeira string de cada comando) a função de leitura identificar uma inconsistência, ela chama a mensagem de erro 19 (Unidade inexistente).

A seguir a lista com o código de erro/sucesso e a mensagem que será impressa no protocolo de saída:

SUCESSOS

- 1 - Rua incluída com sucesso.
- 2 - Rua removida com sucesso.
- 3 - Medição da Cidade realizada com sucesso.
- 4 - Medição do bairro realizada com sucesso.
- 5 - Medição da rua realizada com sucesso.
- 6 - Medição da casa realizada com sucesso.
- 7 - Casa incluída com sucesso.
- 8 - Casa removida com sucesso.
- 9 - Consumo registrado com sucesso.
- 10 - Bairro incluído com sucesso.

ERROS

- 11 - A rua informada não pertence ao bairro.
- 12 - A casa informada não pertence à rua.
- 13 - Já existe uma rua com este id vinculada ao bairro.
- 14 - Já existe uma casa com este id vinculada à rua.
- 15 - Bairro inexistente.
- 16 - O id do bairro não pode ser negativo.
- 17 - O id da rua não pode ser negativo.
- 18 - Ação inexistente.
- 19 - Unidade inexistente.
- 20 - Erro na abertura do arquivo.
- 21 - Já existe um bairro com este id vinculado à cidade.
- 22 - O id da casa não pode ser negativo.
- 23 - O número da casa não pode ser negativo.
- 24 - O consumo não pode ser negativo.
- 25 - Não foi possível alocar memória.

5 RESULTADOS E AVALIAÇÃO

Foram realizados testes utilizando o depurador *Valgrind* com objetivo principal de verificar se o software produzido está administrando a memória alocada de maneira correta. Os resultados foram positivos tanto para casos de sucesso quanto para os diversos erros que possivelmente aconteçam durante a execução. Diversos arquivos de entrada, com comandos estrategicamente errados foram utilizados a fim de se verificar a “desenvoltura” do software.

Um erro identificado, mas não tratado neste projeto é o seguinte:

- Na leitura dos parâmetros (terceiro bloco de informações de acordo com a sintaxe dos comandos), a função de leitura irá procurar um número inteiro. Caso ela não encontre na linha a quantidade de inteiros requeridos, o sistema falha.

Conclui-se que o programa cumpriu o objetivo de realizar o processamento das instruções presentes no arquivo de entrada, relatar possíveis erros encontrados nos comandos e gerar o protocolo de saída com os resultados do processamento.

6 REFERÊNCIAS

CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas. **Introdução a estruturas de dados**: com técnicas de programação em C. 11ª triagem. Rio de Janeiro: Elsevier, 2004.