Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Santa Helena
Bacharelado em Ciência da Computação

Disciplina:	Algoritmos e Estrutura de Dados III		
Alunos:	- Bruno Eduardo Santos Alcantara - José Carlos Seben de Souza Leite - Caio Macedo Lima da Cruz	**R.A.:**	2677156 2651130 2651378
Email:	brunoalcantara@alunos.utfpr.edu.br joseleite@alunos.utfpr.edu.br caiomacedo@alunos.utfpr.edu.br		

Trabalho II - Árvore de Diretórios

Repositório do Trabalho:

https://github.com/BrunoGraveto/ArvoreDeDiretorios

Dificuldades:

As maiores dificuldades que o grupo enfrentou para o desenvolvimento do projeto foi entender como construir a estrutura geral do projeto, de como ficaria a árvore com os filhos e irmãos, além de algumas dificuldades na criação de funções por erros inesperados e coisas do tipo, outra dificuldade que enfrentamos foi na manipulação de strings para a confecção das funções.

Requisitos:

[RF001] - O programa deve oferecer uma interface do tipo linha de comando para o usuário executar operações como: cd, search, rm, list, mkdir, clear, help e exit.

[RF002] - O programa deve obrigatoriamente conter uma função para ler um arquivo in.txt, que deve conter a lista de pastas e arquivos.

[RF003] - O programa deve executar o comando cd <diretório>: * Entra no diretório especificado se ele existir. * Se não existir, imprime as possíveis alternativas. Ex: se o diretório for "Me", deve informar que existe um diretório "Meus Documentos" e "Meus Downloads". * Se não existirem alternativas, imprime "Diretório não encontrado".

[RF004] - O programa deve executar o comando search <arg>: * Busca um arquivo ou pasta pelo seu nome "arg" e informa a sua localização.

[RF005] - O programa deve executar o comando rm <diretório>: * Remove uma pasta e seus arquivos, devendo fazer uma liberação recursiva.

[RF006] - O programa deve executar o comando list: * Lista todos os componentes dentro da pasta atual.

[RF007] - O programa deve executar o comando mkdir <arg>: * Cria uma pasta com o nome "arg" na pasta atual.

[RF008] - O programa deve executar o comando clear: * Limpa o conteúdo da tela imprimindo diversas novas linhas com printf ou usando a chamada de sistema "clear" ou "cls".

[RF009] - O programa deve executar o comando help: * Comando personalizado que deverá explicar quais comandos o programa possui, modo de uso e sua finalidade.

[RF010] - O programa deve executar o comando exit: * Encerra o programa, mas primeiro deve liberar o espaço alocado.

Funções

- **cd:** A função cd recebe o nó atual da árvore (repositório atual) e o diretório no qual se deseja acessar. Com isso, checa se o atual é nulo, ou o caminho. Se não for, verifica se diretório é ".", onde o usuário se mantém no mesmo repositório, ou "..", onde o usuário vai pro pai. Depois das primeiras verificações, a função percorre os filhos do atual (caso não seja nulo, utilizando while), navegando pelos irmãos, até que encontre o filho que possua o caminho desejado. Por fim, retorna o filho desejado.Dessa forma, a função deve ser utilizada para atribuir um novo valor ao no atual.

- **search:** A função search recebe um arg e a raiz como parâmetro assim através de loops ela percorre a árvore toda sempre fazendo o processo de ir para os níveis mais baixos os filhos, depois quando não há filhos ela verifica os irmãos e quando não a nem filhos e nem irmão ela começa a fazer o processo reverso e subir a árvore, até percorrer a árvore toda, ou encontrar o arquivo ou pasta especificado pelo arg e retorna o caminho deste arquivo utilizando um ponteiro auxiliar NO* atual.

- **rm:** A função está recebendo um arg como parâmetro e faz a busca do arquivo/pasta com este nome, ao encontrar ela separa esta pasta "sub-árvore" e realoca os ponteiros, para depois chamar a função auxiliar removeRec para estar fazendo a limpeza da memória recursivamente.

- **list:** A função list é utilizada para listar todos os arquivos e pastas do nó atual (repositório atual). Primeiro, ela verifica se o nó é nulo. Depois, o caminho da atual é verificada, a fim de saber onde estamos. Dessa forma, é utilizado um nó auxiliar, onde seus filhos são percorridos com um loop while, e os nomes dos arquivos/pastas são impressos.

- **mkdir:** Função que cria um nó para uma pasta na árvore a partir do caminho completo informado. Ela extrai o nome, aloca e preenche um novo nó, encontra o pai correspondente na árvore e insere o arquivo como filho desse pai. Retorna 1 se a criação for bem-sucedida e 0 caso contrário.

- **clear:** A função clear verifica os sistema operacional que está sendo utilizado pelo usuário, e executa o comando responsável pela limpeza da interface de linha de comando. Dessa forma, a função funciona em ambos sistemas operacionais.

- **help:** A função exibe uma lista de todos os comandos, utilizando print, onde é explicado ao usuário como ele deve utilizar tais comandos. Para chamar a função

basta digitar help.

- **exit:** A função exit é utilizada para que o programa encerre o loop e as suas atividades, mas também, garante que a memória seja liberada, evitando assim, o vazamento indesejado a mesma. Dessa forma, temos uma função bem estruturada e segura, que garante o funcionamento correto do programa até mesmo para saída.

Extras:

- **uploadArvore:** Função que lê caminhos de um txt e os coloca em uma árvore. Primeiramente, ele verifica se raiz é nula, caso seja retorna 0. Se não, continua e abre o arquivo com o caminho especificado por parâmetro, em seguida ele inicia um loop até o fim do arquivo, verificando linha por linha, a qual ele pega a linha e separa o caminho entre as pastas através do caractere “/”, a qual novamente inicia um loop, verificando se já foram adicionados o arquivo ou pasta um por um, assim se repetindo até o final do arquivo.

- **removeRec:** É uma função auxiliar que está sendo utilizada para fazer remoções recursivas, ela é bem simples e a única coisa que faz quando é chamada e fazer a remoção completa de um NO fazendo essa remoção “de baixo para cima” limpando os filhos, depois que não a filhos limpa os irmãos e quando não há mais irmãos limpa o próprio NO e toda memória alocada.

- **listAll:** Função de list mas nesta está com uma lógica de busca global e não apenas do diretório em que você estiver no cmd, você informa um diretório para a função e ela vai abrir este diretório e printar todos os arquivos deste mesmo diretório, a lógica para percorrer a árvore é a mesma lógica da função de search.

- **rename:** A função rename recebe a raiz do atual e um argumento no formato nome_antigo “nome_novo”. Isso para que seja possível distinguir os dois argumentos a serem utilizados na hora de renomear o arquivo ou pasta. A separação dos nomes é feita da seguinte forma: é criado um buffer de char* para encontrar a primeira aspa, depois o arg é atribuído ao nome_antigo, que corresponderá ao arg até a primeira aspa, depois, o mesmo é feito para segunda aspa, com outro buffer, que a localiza, depois é atribuído o restante da string a nome_novo, ignorando as aspas. É necessário também, eliminar o espaço entre os dois nomes, e isso é feito utilizando loop while em nome_antigo para decrementar caracteres enquanto for espaço. Depois de tratar os nomes, pode-se realizar a troca. Para isso, é utilizada a função search para encontrar o caminho original do arquivo/pasta e um loop while semelhante ao de eliminar espaços é aplicado, mas até encontrar a primeira barra, de trás para frente, removendo assim, o nome antigo

do caminho. Por fim, o NÓ correspondente ao atual é encontrado, o nome é alterado com strdup, a memória do caminho do atual é realocada e o novo local é atribuído, com o novo nome inserido.

- **echo:** A função echo, recebe uma mensagem, que não pode ser nula, e essa mensagem é impressa utilizando a função printf. Caso a mensagem seja nula, ela retorna 0. Caso a mensagem não seja nula, ela é impressa.

- **mkarq:** Função que cria um nó para um arquivo na árvore a partir do caminho completo informado. Ela extrai o nome e a extensão do arquivo, aloca e preenche um novo nó, encontra o pai correspondente na árvore e insere o arquivo como filho desse pai. Retorna 1 se a criação for bem-sucedida e 0 caso contrário.

- **searchPorCaminho:** Faz verificações relacionadas a raiz e o caminho e chama a função recursiva “auxiliar”.

- **buscaCaminhoRecursivo:** verifica se o nó atual é nulo ou se seu caminho corresponde ao buscado. Se não encontrar, procura recursivamente entre os filhos e, se ainda não achar, continua a busca entre os irmãos. Se o nó for encontrado, retorna um ponteiro para ele; caso contrário, retorna NULL.

- **buscaPai:** Função para encontrar o nó pai de um determinado nó. Inicialmente verifica se o caminho (do filho) passado ou a raiz são nulos. Ai através da função strrchr ele salva de trás pra frente a última ocorrência da “‘/”, no caso a última barra, em seguida substituindo-a por “\0”, assim definindo como o final da string(faz isso para retirar o último arquivo/pasta, que seria o filho), assim, por fim chamando a função searchPorCaminho com esse caminho obtido para retornar o nó.

- **inserirNo:** Função auxiliar chamada por mkdir e mkarq, a qual tem a mesma lógica para serem adicionados. Inicialmente, é verificado se o pai é diferente de nulo, caso não seja, o novo nó será inserido como filho desse pai, se ele não tiver filhos, o novo nó se torna o primeiro, caso já existam filhos, o novo nó é adicionado como irmão do último filho. Se nenhum pai for informado, o novo nó é inserido na raiz da árvore. Se a raiz estiver vazia, ele se torna o primeiro nó, caso contrário, é adicionado como irmão do último nó já existente na raiz.

- **liberarArvore:** Função para liberar os nós de toda a Árvore. Inicialmente verifica se a árvore já não está liberada, caso não seja chama a função já feita removerRec, passando como parâmetro a raiz, a qual remove recursivamente todos os nós, e por fim define o ponteiro como nulo.