# **Sistema de Gerenciamento de Idiomas**

#### Alunos: André Santos Gomes e Bernardo Abrahão Mantovani

## Descrição sucinta do Sistema:

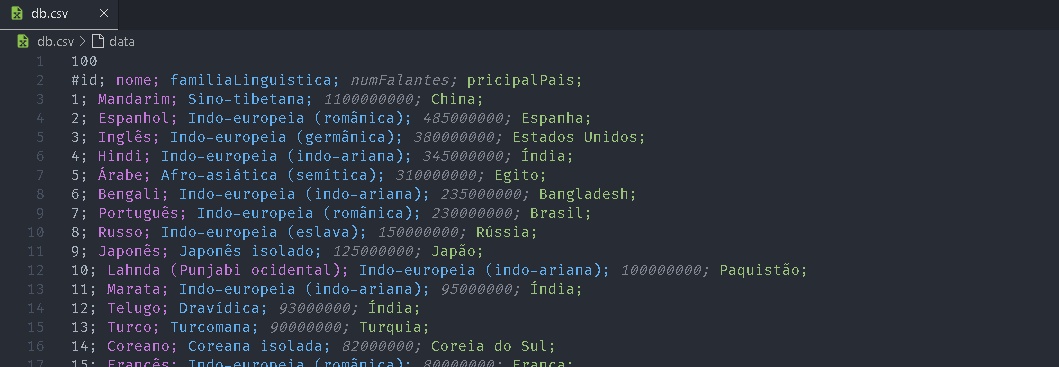
Através de orientação da professora Marluce, produzimos um sistema em C + + capaz de gerenciar dados de idiomas provenientes de um arquivo `.csv`. Dentre as características do programa, temos: capacidade de armazenamento de novos idiomas, atualização deles, exclusão, leitura e ordenação. A capacidade de leitura do sistema inclui também diversos filtros para facilitar a visualização dos dados, desde alteração na quantidade de dados impressos até filtragem de campos específicos. Toda a leitura e escrita de dados é feita com o usuário através do terminal do Linux, onde eles acessam diversos menus através de dígitos numéricos e escolhem o que desejam fazer.

## Como o Algoritmo funciona:

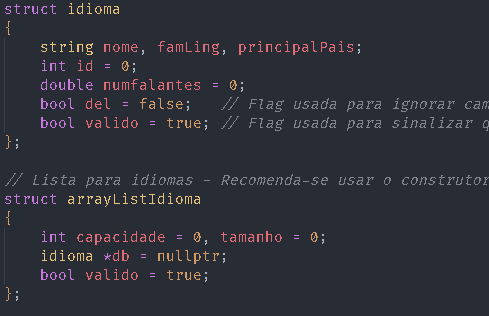
Para melhor clarificação, vamos apresentar o código em etapas:

***Primeira Etapa*** - Inicialização da base de dados

Antes de iniciar a comunicação com o usuário, o programa recebe os dados do arquivo csv através da classe `ifstream` da biblioteca base <fstream>. Primeiramente recebendo o valor da quantidade de linhas na tabela que se localiza no topo do arquivo:



Em seguida pula o cabeçalho, já que ele está ali apenas para facilitar a leitura do usuário, e recebe os dados de cada linha armazenando-os numa pseudo arraylist de idiomas:



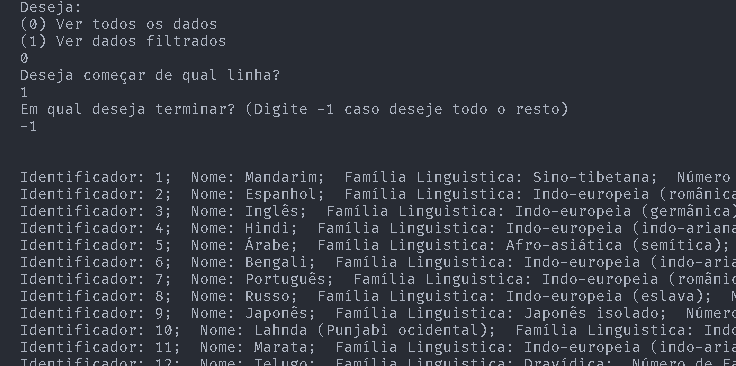
A utilização de uma pseudo arraylist é quase obrigatória para o desenvolvimento desse sistema, já que o tamanho total do vetor e a localização da próximo espaço utilizável dele é essencial para todas as operações feitas nele, e passar esses valores de uma função para a outra iria gerar um malabarismo de parâmetros bastante complexo, dificultando a leitura e manutenção do código.

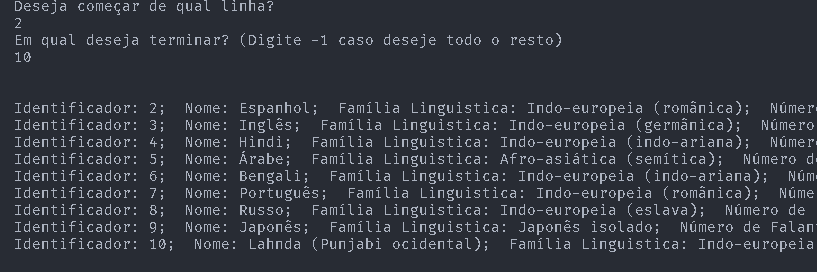
***Segunda Etapa*** - Leitura dos dados no terminal

Após a inicialização do programa terminar, temos um vetor o qual armazena todas as linhas da tabela csv. Em seguida o terminal dá 5 opções para o usuário: 1. Ver dados; 2. Inserir dados; 3. Atualizar dados; 4. Excluir dados; 5. Sair da aplicação. Vamos analisar cada uma dessas opções:

***Terceira Etapa*** - Leitura dos dados do vetor

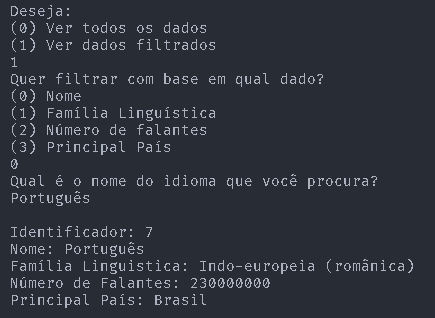
Caso o usuário escolha a opção 1 - Ver dados, ele será apresentado com mais duas opções, Ver todos os dados, a qual simplesmente imprime os dados do array sem filtrá-los, ou então Ver dados filtrados, que mostra mais opções. Toda vez que um conjunto de dados está para ser impresso pelo sistema, você tem a chance de dizer qual faixa do conjunto escolhido você quer acessar, por exemplo: você pode escolher do primeiro elemento até o último (da linha 1 até a linha -1) ou então do segundo elemento até o décimo (da linha 2 até a linha 10).



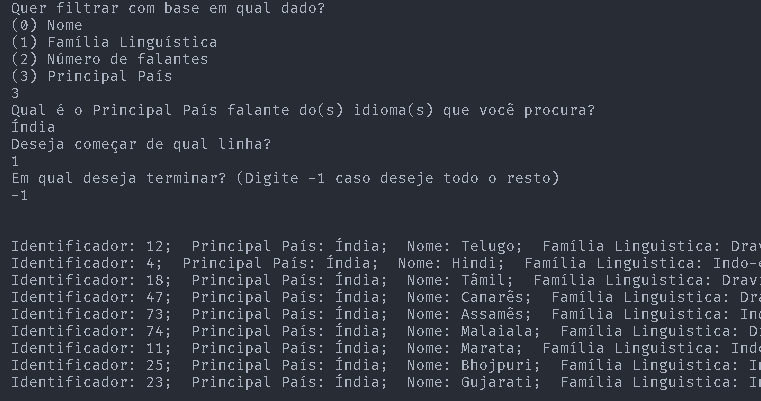


Caso o usuário escolha a segunda opção, ele vai receber as seguintes escolhas: 1. Filtrar os idiomas com base no nome (Ou seja, pesquisar um idioma pelo seu nome); 2. Filtrar os idiomas com base na família linguística deles; 3. Filtrar pelo número de falantes; 4. Filtrar pelo principal país que fala o idioma.

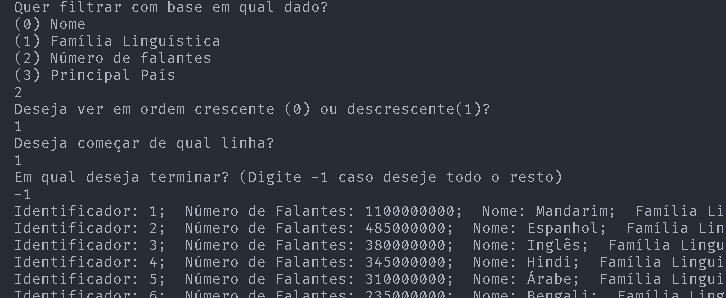
Na primeira opção, o vetor é ordenado através de um algoritmo de quicksort e é feita uma busca binária normal nele. Retornando os dados do idioma que possui aquele nome:



Nas opções 2 e 4, o vetor também é ordenado e é feita uma busca binária um pouco diferente nele: como um país pode possuir vários idiomas e uma família linguística abrange grupos de idiomas, o programa pode precisar imprimir uma faixa de resultados, então a busca binária simples não seria capaz de solucionar esse caso, por isso usamos duas buscas ao invés de uma, uma busca que acha a primeira ocorrência daquele país/família e outra que acha a última ocorrência, assim achamos o início e fim da faixa no vetor que corresponde ao resultado.



Na opção número 3 há apenas a ordenação dos dados, a escolha da ordem da ordenação (crescente ou decrescente) e a escolha da faixa que deseja ver:



Foi implementado dessa forma, uma vez que caso a pessoa queira ver alguma colocação em específico, por exemplo a quinta colocação em ordem crescente, ela pode simplesmente pedir todos os elementos da linha 5 até a linha 5, o que resulta na escrita de um elemento singular.

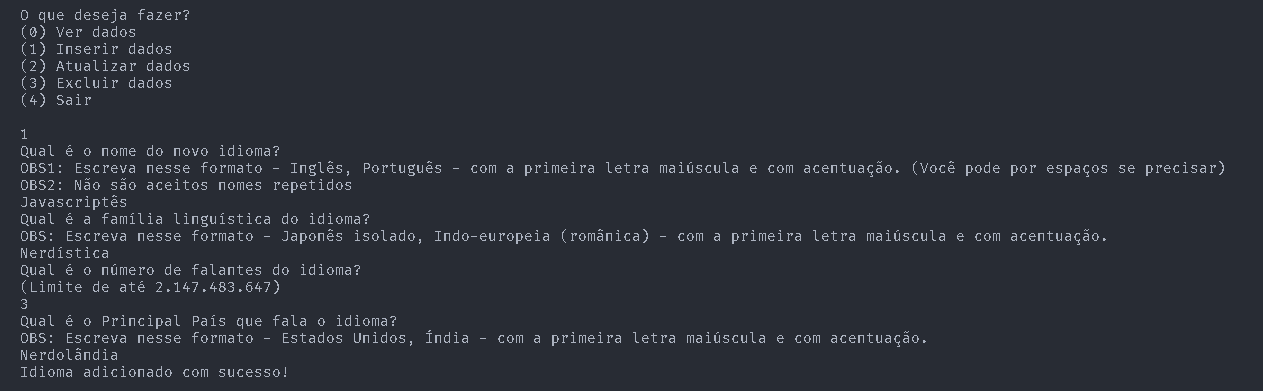
***Quarta Etapa*** - Inserção de novos idiomas

Caso o usuário escolha a opção de Inserção de dados, o sistema irá pedir o nome, a família linguística, o número de falantes e o principal país do novo idioma nesta ordem. Deixando claro a formatação recomendada e algumas observações de entradas inválidas, por exemplo nomes repetidos, as quais o sistema não aceita.

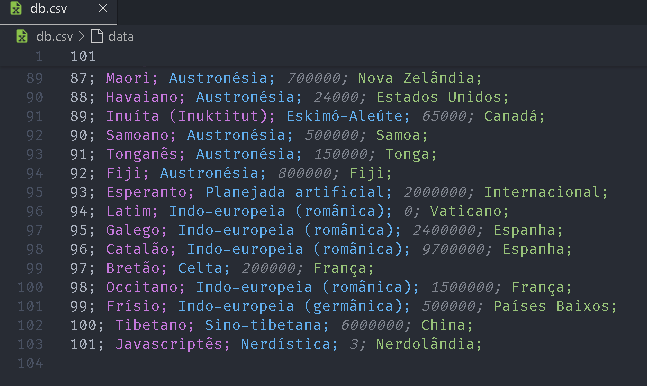
Após os dados serem recebidos, o programa inicia a inserção deles no sistema:

Primeiro os dados que estão separados em diferentes variáveis se juntam num só struct idioma, e logo em seguida passam pelo método `add()` o qual adiciona o novo idioma ao final lógico da arraylist. Caso ela não tenha capacidade para outro idioma, ela passa pelo método resize() o qual aumenta em 10 espaços o seu tamanho e atualiza o valor de `.size`, mas mantém o valor lógico do final do vetor `.at` constante.

Após isso, se a inserção aconteceu com sucesso, o programa confirma ao usuário que o novo idioma foi adicionado.

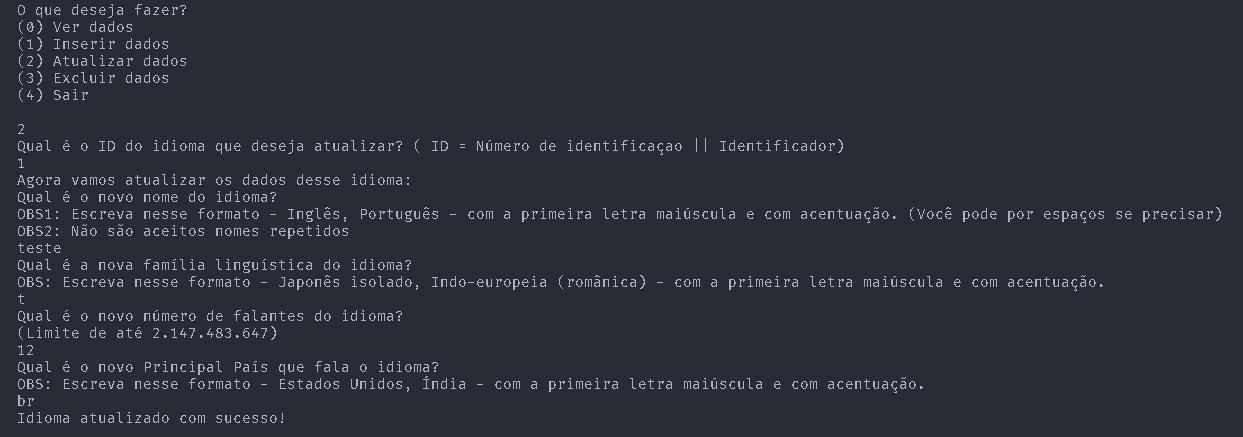


Caso finalizemos o programa, podemos notar o salvamento do novo dado no arquivo csv:

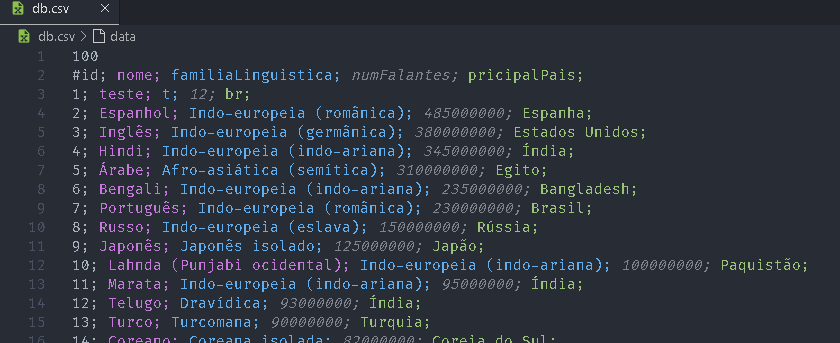


***Quinta Etapa*** - Atualização dos Idiomas

Caso o usuário escolha a opção de atualizar dados, o programa irá pedir o identificador do idioma em específico que deseja atualizar, e em seguida checa a existência dele. Logo após, o sistema pede novos dados de maneira similar à adição de um novo idioma, e confirma ao usuário que a operação pode ser feita, caso não ocorram erros.

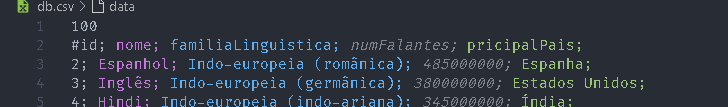


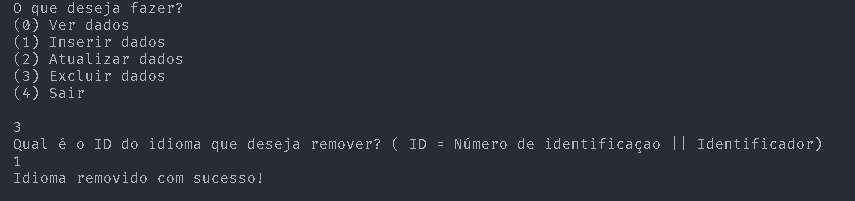
O idioma de ID 1, o qual era mandarim anteriormente, se tornou o novo idioma teste:



***Sexta Etapa*** - Deleção de Idiomas

Caso o usuário escolha remover dados, ele precisará dar ao programa o ID do idioma a ser excluído. Após isso, o programa checará a existência de um idioma com esse ID e em seguida irá marcá-lo para deleção (deixando o atributo `.del` como verdadeiro). Quando um idioma é marcado para deleção, ele não pode ser mais acessado em nenhuma checagem de busca, o sistema automaticamente ignora todas as instâncias com `.del` verdadeiro, portanto esses idiomas não podem ser lidos, atualizados ou removidos novamente. Assim que o programa finaliza, o idioma também é removido do arquivo.





***Sétima Etapa*** - Finalização do programa

Caso o usuário escolha a última opção, a de finalizar a aplicação, o programa em seguida lhe dá 2 escolhas: salvar as alterações feitas, ou descartá-las. Sendo que a primeira opção, logicamente, salva o vetor no arquivo `.csv` enquanto a segunda não faz nada, finalizando a execução.



O os dados são armazenados na ordem apresentada no cabeçalho do arquivo `.csv` dos exemplos anteriores : #id; nome; familiaLinguistica; numFalantes; pricipalPais;

## Observações e Desafios durante o Desenvolvimento:

Esse projeto, apesar de ser simples, requer do aluno diversas habilidades: bom controle de ponteiros, uma boa lógica de programação aplicada, conhecimento das bibliotecas e funções base oferecidas pela linguagem e muita paciência.

Durante o seu desenvolvimento encontramos 4 grandes problemas, sendo eles:

1. Como armazenar os idiomas na memória do sistema de maneira conveniente
2. Como retornar um conjunto de respostas da busca binária ao invés de um elemento só
3. Escolher entre mergesort e quicksort para ordenar os vetores
4. Ignorar os elementos marcados para serem deletados enquanto não eram salvos no arquivo

Para solucionar o primeiro problema, nós utilizamos um struct que mimetiza os atributos de uma classe de lista baseada em arrays, ou seja, um array que pode acrescentar novos elementos além do seu tamanho base e ao mesmo tempo controla seu próprio tamanho, tanto lógico quanto real. Assim pudemos evitar um malabarismo de valores entre as funções, facilitando o acompanhamento das mudanças no valor e a legibilidade do código.

A solução do segundo foi realizar duas buscas binárias, as quais não param enquanto não encontram a primeira e a última aparições de um valor no vetor, assim temos a localização da faixa de valores que correspondem ao pedido do usuário.

No terceiro problema, após estudar a eficiência de cada um dos algoritmos – seus pontos fortes e fracos – chegamos à conclusão de que o quicksort é mais barato na memória, principalmente para arrays, do que o mergesort, apesar de poder precisar de mais operações no pior caso possível.

Para solucionar o quarto problema, não havia um desafio lógico, mas sim um desafio por falta de planejamento, já que percebemos essa necessidade após entrarmos nos estágios finais do desenvolvimento. Então foi árduo atualizar as funções de busca binária e de escrita no terminal para ignorar as instâncias que possuíam `.del` como verdadeiro.

Esses quatro problemas nos ajudaram a entender etapas cruciais na criação de um sistema: precisamos planejar o algoritmo sempre levando em consideração todas as condições impostas pelo enunciado, e não simplesmente executar com base na memória ou na impressão passada pelo problema.

## Conclusão:

O projeto foi uma ótima oportunidade de aprendizado da linguagem C + +, uma vez que precisamos revisar todos os conceitos passados anteriormente no curso e aplicá-los por toda a extensão dele.

No fim, atingimos o objetivo traçado e construímos um sistema de CRUD que salva seus dados num arquivo na memória secundária, carrega os dados de um arquivo csv em um vetor alocado dinamicamente, consegue inserir novos elementos no vetor, remove elementos na memória primária, realiza ordenações e mostra resultados na tela.

O código pode ser acessado juntamente ao relatório no arquivo `.zip`, o qual conta com comentários e com instruções de leitura no arquivo readMe.md.