**COMBSORT**

**O que é, como funciona e complexidade:** O combsort é um algoritmo de ordenação que foi proposto em 1980 por Włodzimierz Dobosiewicz. Ele é uma variação do algoritmo bubblesort, mas é mais eficiente em casos médios e piores.

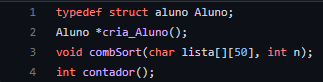
O algoritmo combsort trabalha da seguinte maneira: ele compara elementos que estão separados por uma lacuna (gap) inicialmente grande. A lacuna é reduzida a cada iteração até que se torna 1, no qual o algoritmo é equivalente ao bubblesort.

A vantagem do combsort em relação ao bubblesort é que a lacuna utilizada permite que elementos distantes sejam comparados e trocados. Com isso, o algoritmo é capaz de mover elementos que estão muito distantes do seu local correto de forma mais rápida do que o bubblesort.

O desempenho do combsort é considerado muito bom em comparação com outros algoritmos de ordenação como o mergesort e o quicksort. Ele tem complexidade de tempo O(n log n) no melhor caso e O(n^2) no pior caso, mas na prática ele tende a ser mais rápido do que esses algoritmos em casos médios e piores.

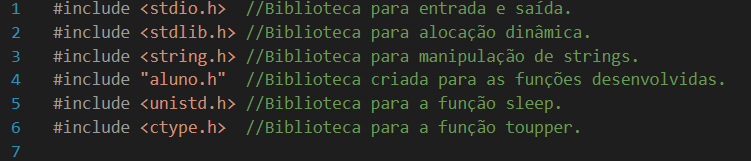
**Documentação do código:**

O primeiro arquivo a ser abordado é, **aluno.h**,o mais simples entre todos, já que tem como função apenas receber os protótipos das funções.



O arquivo possui apenas estas **4 linhas** de código, não existe nenhum ***include*** dentro do mesmo, pois por ser o arquivo de cabeçalho, não vão funções propriamente ditas nele.

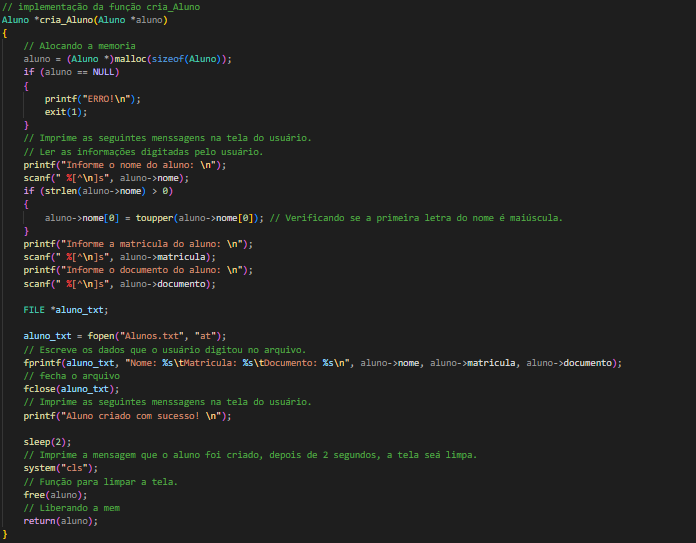
O próximo arquivo se trata do **aluno.c**, que é onde todas as funções serão escritas. O arquivo completo possui **107 linhas** incluindo os comentários. No primeiro momento são incluidas as bibliotecas necessárias e criada a estrutura de alunos.



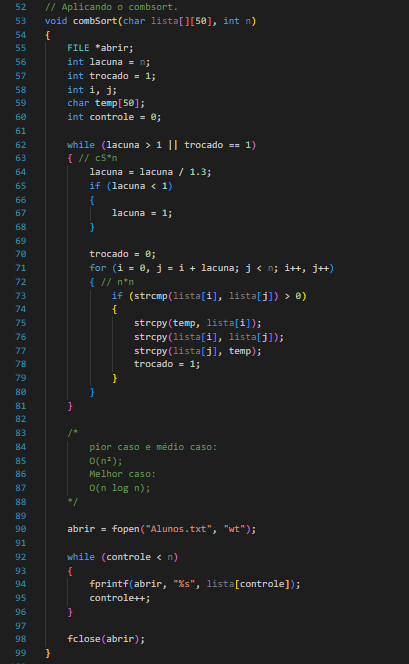
Como pode ser visto o **aluno.h** é incluído neste arquivo, pois é nele que o **aluno.c** se baseia para fazer as funções. Para se definir a estrutura do tipo aluno é usada uma estrutura de formato ***struct aluno{ <elementos da estrutura> }***, neste caso não é necessário usar o comando ***typedef***, pois o mesmo já se encontra no cabeçalho **aluno.h**.

A estrutura do tipo aluno recebe nome, matrícula e documento, todos do tipo caractere, representado por ***char***, pois além de facilitar a organização, também previne as inconsistências do programa, tais como iniciar a matrícula e o documento como 0.

Agora seguindo para a parte em que são atribuídos os elementos dos alunos.

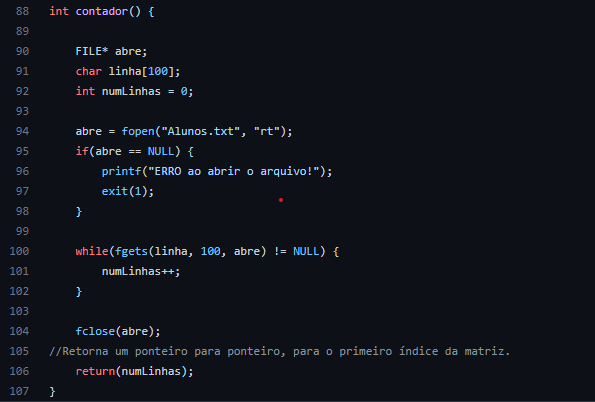
Primeiramente é escrita a função da exata mesma forma que se encontra no **aluno.h**, logo em seguida um aluno é alocado dinamicamente, ocorrendo logo em seguida se a alocação teve êxito, caso não tenha, o programa é encerrado. A seguir são coletados os dados do aluno que serão repassados pelo usuário,os caracteres do nome do aluno ficarão minúsculos devido a função **toupper**. Em seguida é criado a variável do tipo ***FILE***, que indica que é uma variável que se refere a um arquivo. Usando a função ***fopen*** o arquivo é aberto em modo de edição, mais a frente será explicado o motivo de ser aberto assim, logo após usando o ***fprintf*** é escrito no arquivo as informações do aluno. Usando o ***fclose*** o arquivo é fechado e, logo em seguida o terminal do vscode é limpo usando a função system e a função sleep para atardar a execução do programa e permitir que o usuário veja que o aluno foi criado. Depois Aafunção então retorna o aluno.

Agora é aplicado o combsort em si.



Inicialmente são criadas todas as variáveis a serem utilizadas, as principais são a ***lacuna*** e a ***trocado,*** a primeira tem como função receber o total de linhas que serão percorridas pelo algoritmo, a outra é uma variável de controle que verifica se foi feita a troca de dois elementos. É usado um laço de repetição que roda enquanto  ***lacuna*** for maior que 1 ou ***trocado*** ser igual a 1, ou seja, foi efetuada uma troca, logo após a ***lacuna*** é dividida por 1.3, que foi o valor escolhido que mais se encaixa ao que o algoritmo propõe, após isso, se ***lacuna*** for menor do que 1 ela será definida como 1, depois trocado recebe o valor 0, indicando que não houve troca, o programa então entra em outro laço de repetição que faz a parte de trocar os elementos, neste caso, o algoritmo está organizando os alunos por ordem alfabética, assim guardando os elementos em um vetor organizado, ao fim do laço é atribuído 1 para ***trocado,*** se tiver ocorrido a troca.

Após isso é aberto o arquivo no qual os alunos estão em modo de escrita, logo em seguida é escrito no arquivo o vetor de alunos organizados. No fim, o arquivo é fechado.

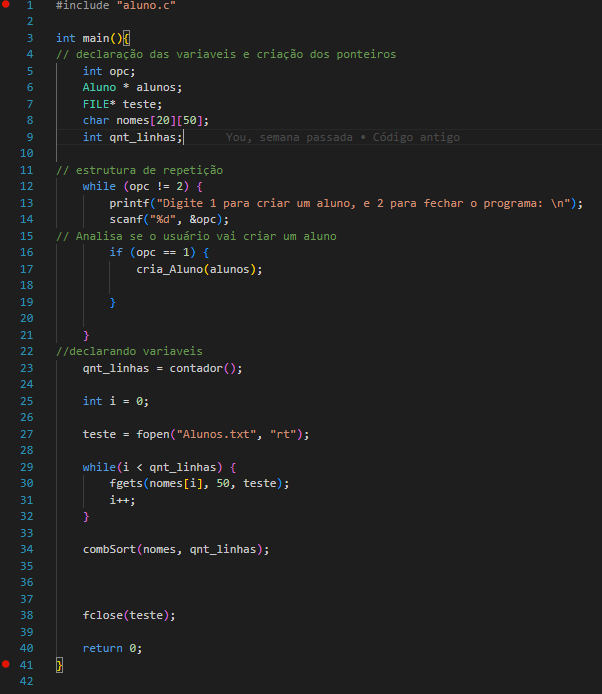
Por fim é feita a função que conta quantas linhas têm o arquivo que contem os alunos, ou seja, quantos alunos têm no arquivo.

Aqui são criadas as variáveis necessárias, em seguida é aberto o arquivo que contém os alunos em modo de leitura, é feita uma verificação se o arquivo foi aberto, caso haja alguma falha o programa é encerrado.

Em seguida é usado um laço de repetição que passa por todas as linhas do arquivo, assim contando elas.

Ao fim da função é retornado a quantidade de linhas do arquivo.

Partindo agora para o último arquivo, vamos para o **main.c** o arquivo onde indica o funcionamento de verdade da aplicação. o arquivo contém ao todo **42 linhas**.



No início são incluídas as bibliotecas a serem utilizadas, sendo uma delas a **aluno.c**, que é onde se encontram as funções. Já dentro da função principal, a ***main,*** primeiro são criadas as variáveis a serem utilizadas.

Logo em seguida é usado um laço de repetição que permite o usuário criar alunos até digitar a tecla **2**, sendo assim, para criar um aluno ele aperta **1**, ao apertar é usada a função ***cria\_Aluno***, após isso é usada a função de ***contador*** para contar quantos alunos foram cadastrados.

Após isso é aberto o arquivo em modo de leitura, que logo em seguida entra em outro laço de repetição para guardar os nomes dos alunos em uma matriz.

Em seguida é aplicado o combsort na matriz preenchida anteriormente. E no fim o programa fecha o arquivo aberto.

**REFERÊNCIAS**

[**ChatGPT**](https://chat.openai.com/chat) **-** utilizado para explicar as funcionalidades e a complexidade do **combsort.**

[**Repositório**](https://github.com/classroom-ufersa/CombSort) **-** Onde o código se encontra.