# Relatório Trabalho 1 de Estrutura de Dados

*Professora: Patrícia Dockhorn Costa*

*Período: 2022/1*

*EDCare: Ed Cuidando de Idosos!*

*Alunos: Rhuan Garcia e Gabriel Braga Ladislau*

# **Introdução:**

Para esse trabalho temos que implementar um compactador e um descompactador de arquivos.

Para isso teremos que mexer no baixo nível de bits, usando o TAD *bitmap*, com ele conseguimos fazer tudo necessário para trabalhar com bytes. Usamos o *doxygen* para documentar o código e conseguir criar um padrão de comentários. Fizemos comentários de todas as funções e estruturas presentes no nosso código. Usamos, também o serviço do *GitHub* para versionar nosso código e trabalhar com mais eficiência.

# **Implementação:**

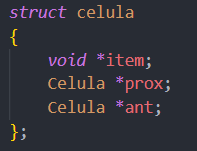
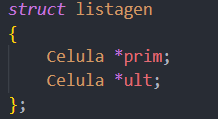
Para começar, fizemos uma implementação sem seguir nenhum padrão, reutilizamos as árvores que implementamos nas aulas para fazer a árvore de Huffman.

Usamos a biblioteca *assert.h* para verificar se os ponteiros usados nas funções do programa estão devidamente alocados e diferentes de *NULL*. Caso esteja *NULL* a função *assert()* para a execução do programa com erro (0 = erro).

Começamos primeiro pelo compactador e seus TADs para depois seguir para o Descompactador.

***TAD ListaGen e ListaArv:***

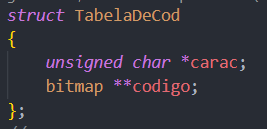
Esse TAD define uma lista genérica duplamente encadeada com sentinela, que no caso é usada para carregar as arvores que serão usadas para depois montar a arvore de *Huffman.* Segue a estrutura da sentinela e da célula da lista respectivamente:



***TAD Codificador:***

O nome é codificador porém este TAD é responsável pela compactação do arquivo.

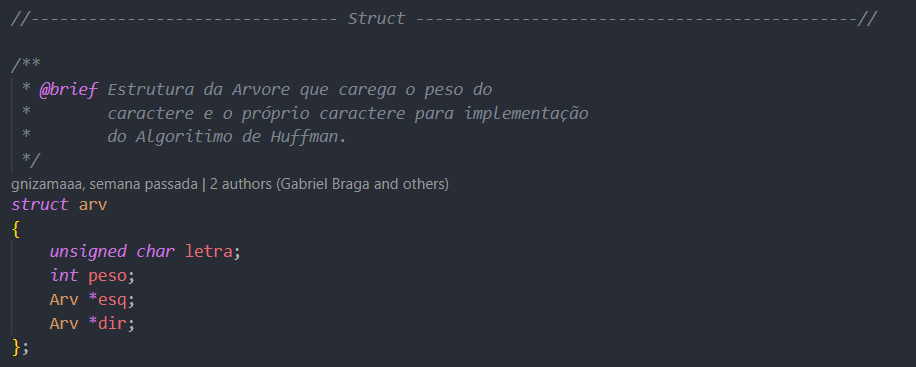
Nele encontramos a estrutura tabela de codificação que carrega para cada *char* em um arquivo, seu correspondente em *bitmap* tendo já a arvore de huffman. Segue abaixo:



Essa Tabela têm suas funções de Montar e Liberar:

***TAD Arvore:***

Para que seja possível implementar a árvore de codificação de Huffman foi necessário que a Arvore em si carregasse um inteiro representando o peso daquele caractere e um *char* que seria o caractere em questão, além da recursividade, onde temos árvore esquerda e direita. Segue a estrutura arv:



No *Arvore.h* temos o *typedef* que define o tipo *Arv* como essa *struct* acima.

Para essa estrutura fizemos 3 funções privadas, *max* usada para calcular a altura da árvore, VarreduraArv que varre a arvore de Huffman para exporta-la para o cabeçalho do arquivo de saída e Recursiva que Percorre a arvore até chegar no caractere requerido adicionando 0's e 1's no bitmap para descobrir como tal caractere está codificado na arvore de Huffman. Segue uma foto abaixo das declarações, iremos mostrar suas definições depois:



Pulando as funções normais de arvore, que não fogem do padrão, seguimos para as mais complexas. Começando pela