

***Programação  
MEEC (2018/2019 – 2º Sem)***

***Projeto Intermédio***



**Grupo: 3**

**Aluno nº: 93030**

**Nome: Bruno Miguel Da Silva Cebola**

**Aluno nº: 93065**

**Nome: Frederico Maria de Almeida Santos**

1. **Introdução**

Em termos gerais, o desafio proposto consiste em desenvolver um programa capaz de adivinhar uma chave secreta do clássico *Mastermind* e de criar e/ou ordenar um registo dos jogo efetuados.

A implementação do algoritmo de jogo tem de seguir diretivas descritas no enunciado fornecido: primeiramente, o algoritmo deve gerar um número definido de tentativas aleatórias; de seguida deve percorrer uma lista implícita de todas a chaves possíveis, testando cada uma com as tentativas anteriores até encontrar uma combinação compatível com estas.

Por outro lado, o “modo de jogo” (teste ou interativo) deve ser escolhido mediante a presença de argumentos (acompanhados pelas suas respetivas flags) ao executar o programa. Estes argumentos configuram a execução do programa no modo de teste.

Por último, o programa não deve simplesmente conseguir criar um registo com todos os jogos efetuados e transcrevê-los para um ficheiro, mas também deve conseguir ler um registo guardado num ficheiro e ordenar os seus os dados de acordo com diferentes critérios, fornecidos pelo utilizador.

1. **Implementação**

Dado que o enunciado promove a tomada de decisões, a nossa interpretação e abordagem do desafio passa pelos seguintes pontos:

* 1. **Descrição das Estruturas de Dados**

Para armazenar os diferentes tipos de dados necessários à execução do programa, foram utilizadas diversas estruturas de dados:

* **dados:** estrutura composta por 3 variáveis do tipo int (tempo, tentativas e vitoria). É usada para guardar os melhores dados de jogo para o modo torneio do projeto intermedio.
* **letras:** estrutura composta por 1 variável do tipo char (letra) e 1 ponteiro para outra estrutura do tipo letras (\*next). É usada para criar a lista de cores necessária ao algoritmo que adivinha a chave de forma autónoma.
* **tentativas:** estrutura composta por 2 variáveis do tipo char (\*tentativa e resultado[5]), 3 variáveis do tipo int (tent\_ID, pretas e brancas) e 2 ponteiros para outra estrutura do tipo tentativas (\*next e \*prev). É usada para criar a lista de tentativas efetuadas nos diversos modos de jogo do programa.
* **defs:** estrutura composta por 1 variável do tipo char (repetição\_cores) e 7 variáveis do tipo int (num\_jogadores, duração\_jogo, num\_jogos, num\_cores, tamanho\_chave, tentativas e tentativas\_alea). É usada para guardar as definições de jogo.
* **flags:** estrutura composta por 3 variáveis do tipo int (init, hist e ord). É usada para guardar a posição do vetor argv onde cada ficheiro se encontra.
* **game\_reg:** estrutura composta por 4 variáveis do tipo int (game\_ID, colors, key\_size e tentativas), 1 variável do tipo float (game\_time), 4 variáveis do tipo (repet, player\_ID[5], \*key, \*player\_name), 2 ponteiros para outra estrutura do tipo game\_reg (\*next e \*prev) e 1 ponteiro para uma estrutura do tipo tentativas (\*first). É usada para criar a lista do registo de jogo, usada para realizar a ordenação do ficheiro de histórico.
* **hist\_data:** estrutura composta por 2 variáveis do tipo int (ID, player\_ID) e 1 ponteiro para uma estrutura do tipo game\_reg (\*last). É usada para guarda os maiores valores encontrados no ficheiro de histórico.
* **DEFAULT\_FILE:** macro que guarda o nome do ficheiro de histórico usado como predefinição, caso não seja passado nenhum pela linha de comandos.

Para além destas estruturas de dados, utilizamos ainda outras variáveis para, por um lado, guardarem e processarem os inputs do utilizador e, por outro, auxiliarem na correta execução do programa (como variáveis locais às funções).

* 1. **Descrição da Organização dos Ficheiros e das Funções**

Com o intuito de deixar o ficheiro principal o mais limpo e percetível possível, decidimos separar as funções em diversos ficheiros .c e .h, que serão de seguida explicados.

* + 1. Ficheiros game:
* **mode\_check**: função utilizada primariamente para determinar o modo de jogo. De forma secundária, chama ainda a função (**test\_mode\_config**) que preenche a estrutura cmd\_flags com a posição no array argv onde se encontra o nome de cada ficheiro.
* **modo\_ordenacao**: função geral que chama as funções que realizam a ordenação do histórico de jogo.
* **modo\_auto**, **modo\_inter\_pc** e **modo\_inter\_intermedio**: funções gerais que chamam as funções que permitem realizar cada modo de jogo.
  + 1. Ficheiros key:

Library que contem as funções que permitem a implementação do algoritmo que descobre a chave de forma autónoma.

* + - * **listaCores**: função que cria o array de listas que contem todas as cores possíveis para cada posição da chave.
      * **tentativas\_alea**: função geral que cria as tentativas aleatórias. No seu corpo, chama 2 funções auxiliares: **fillAlea**, que gera, avalia e insere a tentativa aleatória na lista de tentativas, e **verificaResultAlea**, que otimiza o algoritmo retirando cores da lista de cores com base na avaliação (PxBy) da tentativa.
      * **keyFinder**: função geral que realiza a iteração pelas diversas tentativas possíveis e encontra a chave de jogo. Chama, no seu interior, 3 funções auxiliares quando a tentativa atual é compatível com todas as tentativas anteriores: **fillLogic**, que avalia a tentativa e a insere na lista de tentativas, **verificaResultLogic**, que otimiza o algoritmo retirando cores da lista de cores com base na avaliação (PxBy) da tentativa, e **reset**, que permite iterar por todas as cores.
    1. Ficheiros sort:

Library que contem as funções que permitem a implementação do algoritmo de ordenação do ficheiro de histórico.

* + - * **recursive\_bubble\_sort\_fast** e **recursive\_bubble\_sort\_short**: funções recursivas que realização a ordenação do ficheiro de histórico de acordo com os parâmetros fast/short. Para tal, iteram a lista, comparando os elementos consecutivos entre si, alterando a sua posição caso estejam mal ordenados.
      * **reord\_2\_elements**: função que permite reordenar dois elementos consecutivos de uma lista.
      * **save\_game\_ini**, **save\_ key** e **save\_guess\_ini**: funções que permitem criar o ficheiro de histórico no modo interativo do projeto intermédio.
    1. Ficheiros files:
       - **read\_init**: função que lê o ficheiro das inicializações e preenche a estrutura defs com as definições de jogo.
       - **read\_hist**: função que lê o ficheiro de histórico e cria a lista com o registo de jogo.
       - **write\_file\_raw** e **write\_file**: funções utilizadas para escrever dados no ficheiro de registo, ao longo do jogo ou através da lista com o registo de jogo, respetivamente.
    2. Ficheiros memory:
       - **free\_game\_registry** e **free\_guess\_list**: funções utilizadas para libertar a memória alocada para criar a lista do registo de jogo.
       - **clear\_memory** e **clear\_keyFinder**: funções para libertar a memória alocada durante os diversos modos de jogo (modos interativos e modo teste).
    3. Ficheiros intermedio:

Library que contem as funções que permitem implementar o modo interativo 1, referente ao projeto intermédio.

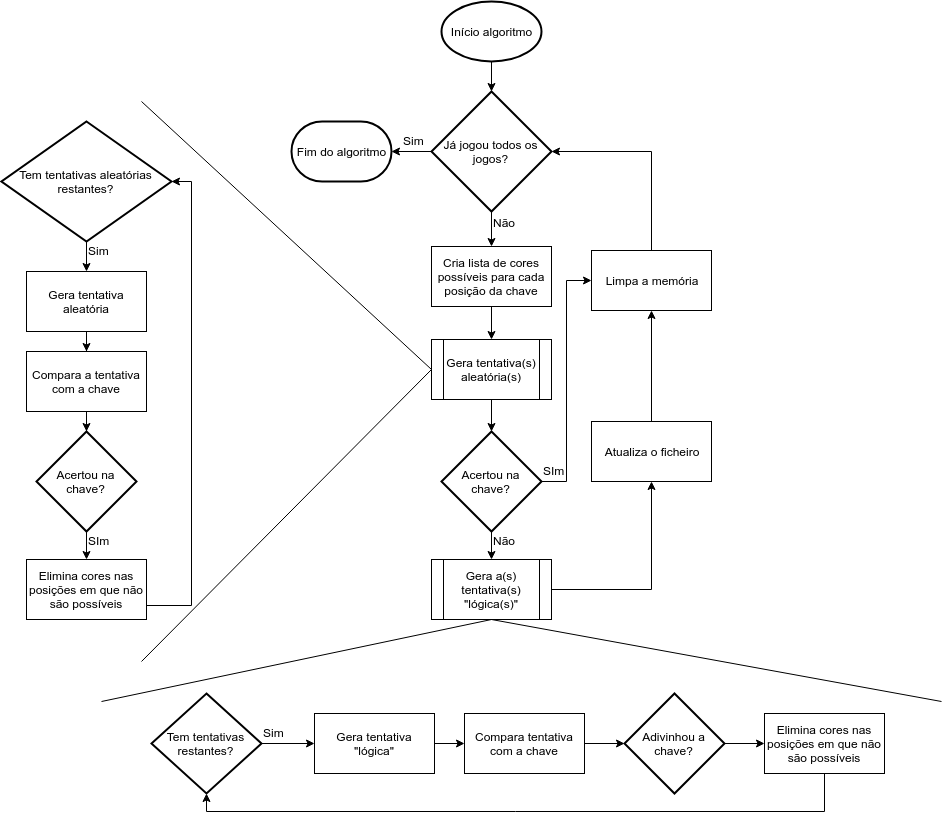
* 1.  **Fluxograma do Algoritmo keyFinder**

Fig 1. Algoritmo KeyFinder

* 1. **Fluxograma do Algoritmo de Ordenação**

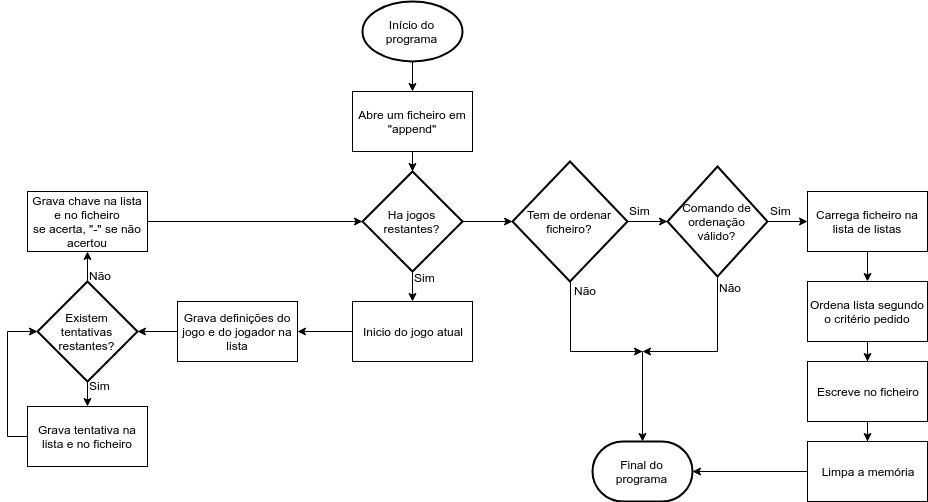


Fig 2. Algoritmo de Ordenação

1. **Conclusão**

Em suma, para além de desenvolver um programa que se limita cumprir as exigências do enunciado, esteve sempre presente a vontade de otimizar ao máximo o código em termos de utilização de memória e rapidez de execução.

Por outro lado, um objetivo importante foi deixar o código fácil e confortável de ler ao utilizar nomes de variáveis facilmente identificáveis e ao apostar fortemente no uso comentários tanto ao longo do código como antes de cada função. Por outro lado, com o intuito de verificar a inexistência de erros e *warnings* adicionais, compilamos o nosso programa com as seguintes *flags*: *-Wall -Wextra -ansi -pedantic -std=c99;* e adicionalmente, também corremos o programa com a ferramenta de depuração Valgrind (valgrind --leak-check=full)corrigindo os erros e *warnings* encontrados.