

***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**

AAAAA

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 outras variáveis para, por um lado, guardarem e processarem os inputs do utilizador (nomeadamente as configurações) e, por outro, auxiliarem na correta execução do programa (como variáveis ligadas ao tempo).

* 1. **Descrição das Funções Principais**

Com o objetivo de deixar a nossa função *main* mais curta, simples e clara, recorremos à utilização de múltiplas funções que abaixo descrevemos:

* **introducao:** função que dá as boas-vindas ao(s) utilizador(es) e que imprime as regras fundamentais do jogo.
* **cleanslate:** função utilizada para absorver qualquer *input* extra ou inesperado do utilizador de forma a evitar o vazamento deste para a próxima *input query*, ou a geração de erros de execução.
* **clearScreen:** função que limpa o terminal após o utilizador clicar no *enter.*
* **countdown:** função que inicia uma contagem decrescente e apaga todo o *input* do *stdin* durante a sua execução.
* **initialization:** tanto esta função como as suas variantes (initializationNames e initializationRepetitions) são utilizadas para inicializar e definir os parâmetros do jogo.
* **checkCombinação:** função que verifica se a combinação dos parâmetros tamanho\_chave, numero\_cores e repeticao\_cores é válida.
* **createKey:** função que gera uma chave secreta e aleatória segundo os parâmetros definidos durante as inicializações.
* **userAttempt:** função para confirmar a validade da jogada efetuada pelo jogador. Ou seja, verifica que o tempo não acabou e que a chave tem o tamanho certo.
* **checkInput:** função que verifica que os inputs dos jogadores durante o decorrer do jogo são válidos (i.e., são jogadas possíveis). Esta é utilizada dentro da função userAttempt.
* **comparaChave:** função para comparar a chave de jogo com a jogada feita pelo jogador e imprime o número de pinos brancos e pretos.
* **jogo:** função que realiza o torneio (todos os jogos de todos os jogadores), chamando diversas funções: createKey, userAttempt e comparaChave.
* **criaDados:** função que gera os dados de jogo gerais de cada jogador como a média de tempo de jogo e o número de vitórias.
* **vencedor:** função que determina o vencedor de acordo com os dados dos jogos do torneio. A sua variante (resultados) determina os vencedores nas duas outras categorias.
* **ShowData:** função que apresenta a performance de cada jogador caso seja pedido.
  1. **Fluxograma Geral**
  2. **Fluxograma das Funções Principais**

1. **Conclusão**

Em suma, para além de desenvolver um programa que se limita a cumprir as exigências do enunciado, esteve sempre presente a vontade de implementar características suplementares que melhorariam a experiência dos utilizadores/jogadores.

Por outro lado, tivemos sempre o cuidado deixar o código percetível, pelo que utilizámos nomes de variáveis autoidentificáveis e apostámos fortemente no uso comentários ao longo do código, principalmente antes de cada função.

Adicionalmente, e com o intuito de verificar a inexistência de erros e *warnings* adicionais, compilámos o nosso programa com as seguintes *flags*: *-Wall -Wextra -ansi -pedantic -std=c99;* e corremos ainda o programa com a ferramenta de depuração Valgrind (valgrind --leak-check=full)corrigindo todos os erros e *warnings* encontrados.