

## AULA PRÁTICA 02

- **Data de entrega: Até 18 de dezembro às 23:59:59.**

- **Procedimento para a entrega:**

1. Submissão: via **run.codes**.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos `.h` e `.c` sempre que cabível.
5. Os arquivos a serem entregues, incluindo aquele que contém `main()`, devem ser compactados (`.zip`), sendo o arquivo resultante submetido via **run.codes**.
6. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.
7. Siga atentamente quanto ao formato da entrada e saída de seu programa, exemplificados no enunciado.
8. Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.
9. A avaliação considerará o tempo de execução e o percentual de respostas corretas.
10. Eventualmente, serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação.
11. Considere que os dados serão fornecidos pela entrada padrão. Não utilize abertura de arquivos pelo seu programa. Se necessário, utilize o redirecionamento de entrada.
12. Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.
13. Códigos cuja autoria não seja do aluno, com alto nível de similaridade em relação a outros trabalhos, ou que não puder ser explicado, acarretará na perda da nota.
14. Códigos ou funções prontas específicos de algoritmos para solução dos problemas elencados não são aceitos.
15. Não serão considerados algoritmos parcialmente implementados.

- **Bom trabalho!**

## Ajudante do Coelho da Páscoa

No ano de 2023, o Coelho da Páscoa não poderá sair de casa para entregar os ovos de páscoa por ter quebrado as quatro patas em um acidente envolvendo chocolate e lutas de boxe. Então ele pediu emprestado os duendes do papai noel para que eles fossem entregar os ovos de chocolate no seu lugar no dia da páscoa.

Como eles são bastante inexperientes, principalmente sobre ovos de chocolate, irão se dividir em vários times compostos de quatro membros: um líder, um confeiteiro, um entregador e um embrulhador. O plano do Coelho da Páscoa é que os líderes das equipes sejam sempre os duendes mais velhos. Por esse motivo, ele pediu para todos os duendes escreverem seus nomes e idades em uma lista. Como você é um duende programador, e já ajudou o Papai Noel a organizar uma lista de times semelhante uma outra vez, você também resolveu ajudar o Coelho da Páscoa.

Seguem abaixo algumas regras e fatos:

- A lista deve ser organizada em ordem decrescente de idade;
- Não existem dois duendes com a mesma idade;
- Não existem dois duendes de mesmo nome;
- Nenhum duende tem mais de 20 caracteres em seu nome;

- Os duendes da lista têm idade entre 10 e 200 anos;
- O primeiro 1/4 dos duendes (os mais velhos), serão os líderes dos times;
- O primeiro 1/4 dos duendes, serão os confeitores dos times;
- O primeiro 1/4 dos duendes, serão os entregadores dos times;
- O último 1/4 dos duendes serão os embrulhadores dos times.

Exemplo: Se há 8 duendes na lista, haverá dois times, onde o duende mais velho é líder do time 1, e o segundo mais velho é líder do time 2. O terceiro mais velho é confeiteiro do time 1 e o quarto mais velho é o confeiteiro do time 2. O quinto é o entregador do time 1 e o sexto é o entregador do time 2. O sétimo mais velho é o embrulhador do time 1 e o último mais velho é embrulhador do time 2

Especifique no **main** em forma de comentário, qual a complexidade das funções: `escalarTimes` e `proximoMaisVelho`.

## Considerações

- Não altere o nome dos arquivos.
- O arquivo `.zip` deve conter na sua raiz somente os arquivos-fonte.
- Há vários casos de teste. Você terá acesso (entrada e saída) de casos específicos para realizar os seus testes localmente.

O código-fonte deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. A informação de cada duende deve ser armazenada em um tipo abstrato de dados criado especificamente para isso, contendo campos **nome**, **idade** e **escalado**. O último campo deste tipo abstrato de dados deve ser utilizado para determinar se o duende já foi escalado para um time ou não. Todas as informações dos duendes de um caso de teste devem ser armazenadas em um vetor alocado dinamicamente (e posteriormente liberado) para cada caso de teste.

Os dados de um time devem ser armazenados em um tipo abstrato de dados criado especificamente para isso, contendo campos `nomeLider`, `nomeConfeiteiro`, `nomeEntregador` e `nomeEmbrulhador` para os nomes, além de campos `idadeLider`, `idadeConfeiteiro`, `idadeEntregador` e `idadeEmbrulhador` para as idades. Os dados dos times devem ser armazenados em um vetor alocado dinamicamente (e posteriormente liberado) para cada caso de teste.

Note que **não** se deve utilizar algum algoritmo de ordenação para resolver este problema. Basta ir selecionando qual o próximo duende mais velho e seguir escalando os times.

## Especificação da Entrada e da saída

A entrada é composta de um número inteiro  $n$ , múltiplo de quatro, que representa a quantidade de duendes na lista (não é necessário validar o valor de  $n$ , ele sempre será um múltiplo de quatro). Em seguida as próximas  $n$  linhas contém o nome e a idade de cada duende.

A saída é composta de cinco linhas por time.

1. A primeira linha deve seguir o formato “TIME X”, onde X é o número do time.
2. A segunda linha deve seguir o formato “PILOTO > ” seguido pelo nome e idade do líder.
3. A terceira linha deve seguir o formato “CONFEI > ” seguido pelo nome e idade do confeiteiro.
4. A quarta linha deve seguir o formato “ENTREG > ” seguido pelo nome e idade do entregador.
5. Por fim, a quinta linha deve seguir o formato “EMBRUL > ” seguido pelo nome e idade do embrulhador.

Depois de cada time, deverá haver uma linha em branco, inclusive após o último time.

## Diretivas de Compilação

```
$ gcc -c duende.c -Wall
$ gcc -c pratica.c -Wall
$ gcc duende.o pratica.o -o exe
```

Entrada	Saída
12 Kepeumo 67 Necoi 62 Sindri 21 Seies 77 Ciule 49 Gyun 99 Chico 25 Finron 27 Norandir 66 Galvaindir 55 Pinhuobor 70 Shadow 23	TIME 1 PILOTO > Gyun 99 CONFEEI > Kepeumo 67 ENTREG > Galvaindir 55 EMBRUL > Chico 25  TIME 2 PILOTO > Seies 77 CONFEEI > Norandir 66 ENTREG > Ciule 49 EMBRUL > Shadow 23  TIME 3 PILOTO > Pinhuobor 70 CONFEEI > Necoi 62 ENTREG > Finron 27 EMBRUL > Sindri 21

### Avaliação de *leaks* de memória

Uma forma de avaliar se não há *leaks* de memória é usando a ferramenta valgrind. Um exemplo de uso é:

```
gcc -g -o exe *.c -Wall; valgrind --leak-check=yes -s ./exe < casoteste.in
```

Espera-se uma saída com o fim semelhante a:

```
==38409== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Para instalar no Linux, basta usar: `sudo apt install valgrind`.