

Aula Prática I - TADs

- Procedimento para a entrega:

1. Submissão: via **Moodle**.
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 **Moodle**.
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!**

Estoque de Pokémons

Crie um programa que leia uma lista de pokémons, com nome, tipo e pontos de combate, indicando um estoque de pokémons e mostre o pokémon com maior PC por tipo e todos os pokémons repetidos.

Especificação da Entrada e da Saída

Seu algoritmo deve ler uma sucessão de linhas, cada uma com três valores *nome*, *tipo* e *pc*, onde *nome* é uma *string* que indica o nome do pokémon, *tipo* é uma *string* que representa o tipo do pokémon e *pc* um inteiro para indicar os pontos de combate. A leitura deve encerrar quando o usuário digitar 0.

A saída do programa deve imprimir na tela o pokémon com maior PC para cada tipo listado na entrada e também todos os pokémons que são repetidos no estoque.

Entrada	Saída
<pre>abra psiquico 770 bulbassauro planta 650 magikarp agua 152 lapras agua 860 slowpoke psiquico 699 magikarp agua 300 0</pre>	<pre>Maior PC por tipo: psiquico: abra planta: bulbassauro agua: lapras Repetidos: magikarp</pre>

Siga o protótipo fornecido, sem alterar os nomes dos arquivos e das funções principais e execute sua implementação com os testes disponibilizados, comparando a saída esperada da sua saída.

Diretivas de Compilação

As seguintes diretivas de compilação devem ser usadas (essas são as mesmas usadas no run.codes).

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

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:

```
1 gcc -g -o exe *.c -Wall
2 valgrind --leak-check=full -s ./exe < casoteste.in
```

Espera-se uma saída com o fim semelhante a:

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

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