

**TABELA HASH**

- **Submissão via Moodle.**

- **Data e hora de entrega disponíveis no Moodle.**

- **Procedimento para a entrega:**

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

- **Bom trabalho!**

A Biblioteca Mágica

Imagine que você foi contratado por uma biblioteca mágica secreta, onde livros de diferentes universos são armazenados. Cada vez que um novo livro é adicionado, ele contém palavras-chave mágicas que precisam ser catalogadas para que os bibliotecários possam encontrá-las facilmente no futuro. No entanto, essas palavras mágicas são poderosas e precisam ser armazenadas em uma Tabela *Hash* especial, de modo que nenhuma palavra mágica se perca ou cause uma colisão mágica desastrosa.

Sua tarefa, como o novo Guardião da Tabela *Hash*, é implementar essa Tabela *Hash* para garantir que cada palavra mágica seja armazenada de maneira eficiente e segura. Ela utiliza a função de transformação $H(\cdot)$ para acessar a *hash* abaixo:

$$H(s) = \left(\sum_{i=0}^{|s|-1} s[i] \times p[i \bmod |p|] \right) \bmod M,$$

onde, s representa a palavra mágica (ou chave) que precisa ser armazenada, p é um vetor de pesos mágicos fornecido pelos antigos guardiões, e M é o tamanho máximo da Tabela *Hash*. O símbolo $|s|$ representa o tamanho da palavra mágica, $|p|$ o tamanho do vetor de pesos, e \bmod é a operação de resto, garantindo que o resultado da função de transformação esteja dentro dos limites da Tabela *Hash*.

Como a magia é sensível, seu programa deve ser “*CaSe InSeNsItIvE*”. Isso significa que as palavras “*Apple*”, “*apple*”, e “*APPLE*” devem ser tratadas como a mesma palavra mágica. Além disso, caracteres não-alfabéticos, que podem perturbar a magia, devem ser ignorados, ou seja, você deverá desconsiderar qualquer caractere que não seja uma letra (‘a’ a ‘z’ e ‘A’ a ‘Z’).

E tem mais: como essas palavras mágicas têm energia poderosa, às vezes elas podem colidir ao serem armazenadas. Seu programa deve utilizar a abordagem de **Endereçamento Aberto** para resolver essas colisões, garantindo que todas as palavras mágicas sejam armazenadas corretamente sem destruir os encantamentos dos livros.

Sua missão é implementar o TAD *Hash* que vai garantir a segurança e organização do conhecimento mágico contido na biblioteca. Que a força dos guardiões esteja com você!

Considerações

O código-fonte deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. Uma Tabela *Hash*, preenchida e percorrida para determinação da solução.

- 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.

Especificação da Entrada e da saída

A entrada será dada por vários casos de teste. A primeira linha possui um inteiro N , indicando o número de casos de teste. Cada caso de teste ocupa duas linhas da entrada. A primeira linha é iniciada por dois inteiros, M e P , indicando respectivamente o tamanho da Tabela *Hash* e o tamanho do vetor de pesos. Em seguida, ainda na primeira linha, são apresentados P números inteiros, representando os valores do vetor de pesos. A segunda linha é iniciada por um inteiro S , $S < M$, de chaves que serão inseridas na Tabela *Hash*. Em seguida as S chaves são apresentadas, separadas por um espaço.

A saída de cada caso de teste deve ocupar apenas uma linha, contendo o inteiro que representa a posição em que o último item foi inserido. Observe que se uma chave repetida for inserida, você deve informar -1.

Entrada	Saída
2 10 5 1 2 3 4 5 2 exercicio pratico 10 1 1 5 galo atletico campeonato versus cruzeiro	1 5

Entrada	Saída
2 10 5 1 2 3 4 5 3 exercicio exercicio pratico 10 1 1 6 galo atletico campeonato versus cruzeiro galo	1 -1

Diretivas de Compilação

```
$ gcc -c hash.c -Wall  
$ gcc -c pratica.c -Wall  
$ gcc hash.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:

```
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`.