Nome: Marcos Antonio Carlos Garcia

RA: 840121

TRABALHO DE ESTRUTURA DE DADOS

```c
int main() {

    Aluno* alunos[MAX_ALUNO];

    FILE* arq = ler_arquivo("dados.txt");

    if (arq == NULL) printf("Arquivo nao existe.\n");

    ler_conteudo_arquivo(arq, alunos);

    Hash* hash = criar_tabela_hash();

    for (int i = 0; i < MAX_ALUNO; i++) {
        add(hash, alunos[i]);
    }

    int acumulador = 0;
    int ociosa = 0;
    for (int i = 0; i < MAX_LISTA; i++) {
        Aluno* aluno = hash->lista[i];
        if (aluno == NULL) {
            ociosa++;
        }
        int count = 0;
        while(aluno) {
            count++;
            aluno = aluno->prox;
        }
        printf("Quatidade de elementos na posicao %d da hash-table: %d\n", i, count);
        acumulador += count;
    }
    printf("------------------------------------------------------------------------\n");
    printf("Media de elementos nas listas: %d\n", acumulador/MAX_LISTA);
    printf("------------------------------------------------------------------------\n");
    printf("Colisoes: %d\n", get_colisoes());
    printf("------------------------------------------------------------------------\n");
    printf("Posicoes ociosas na tabela hash: %d\n", ociosa);
    printf("------------------------------------------------------------------------\n");

    limpesa_de_memoria(hash, arq);

}
```

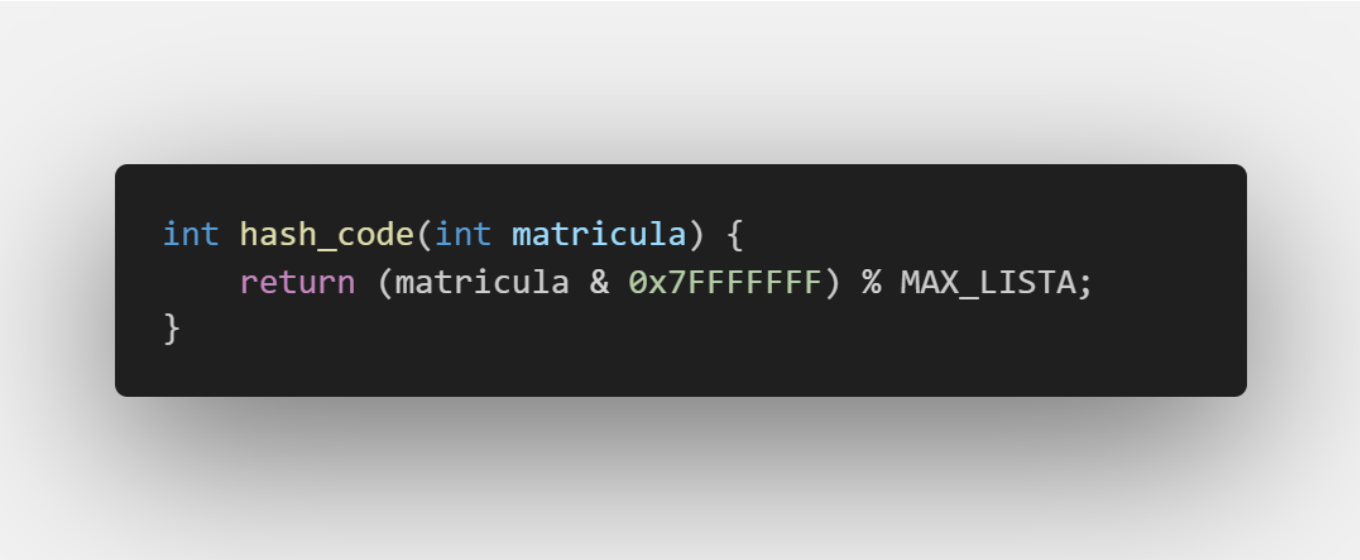
Começamos com a leitura do arquivo e a construção das estruturas com os dados e a alocação das estruturas em um vetor de mesmo tipo. Logo após isso fazemos a criação da estrutua "Hash" e a alocação dos dados nessa estrutura. Por fim temos uma ação para recuperar as informações sobre o processo.

1 Criação da estrutura Hash

```c
// main.c
Hash* hash = criar_tabela_hash();
// =====

// hash_table.h
#define MAX_LISTA 1000

typedef struct {
    Aluno* lista[MAX_LISTA];
} Hash;

Hash* criar_tabela_hash();
int hash_code(int matricula);
void add(Hash* hash_table, Aluno* aluno);
int get_colisoes();
// =====

// hash_table.c
Hash* criar_tabela_hash() {
    Hash* hash = (Hash*) malloc(sizeof(Hash));
    if (hash == NULL) {
        printf("Erro ao criar a tabela Hash.\n\n");
        exit(0);
    }
    for (int i = 0; i < MAX_LISTA; i++) {
        hash->lista[i] = NULL;
    }
    return hash;
}
// =====
```

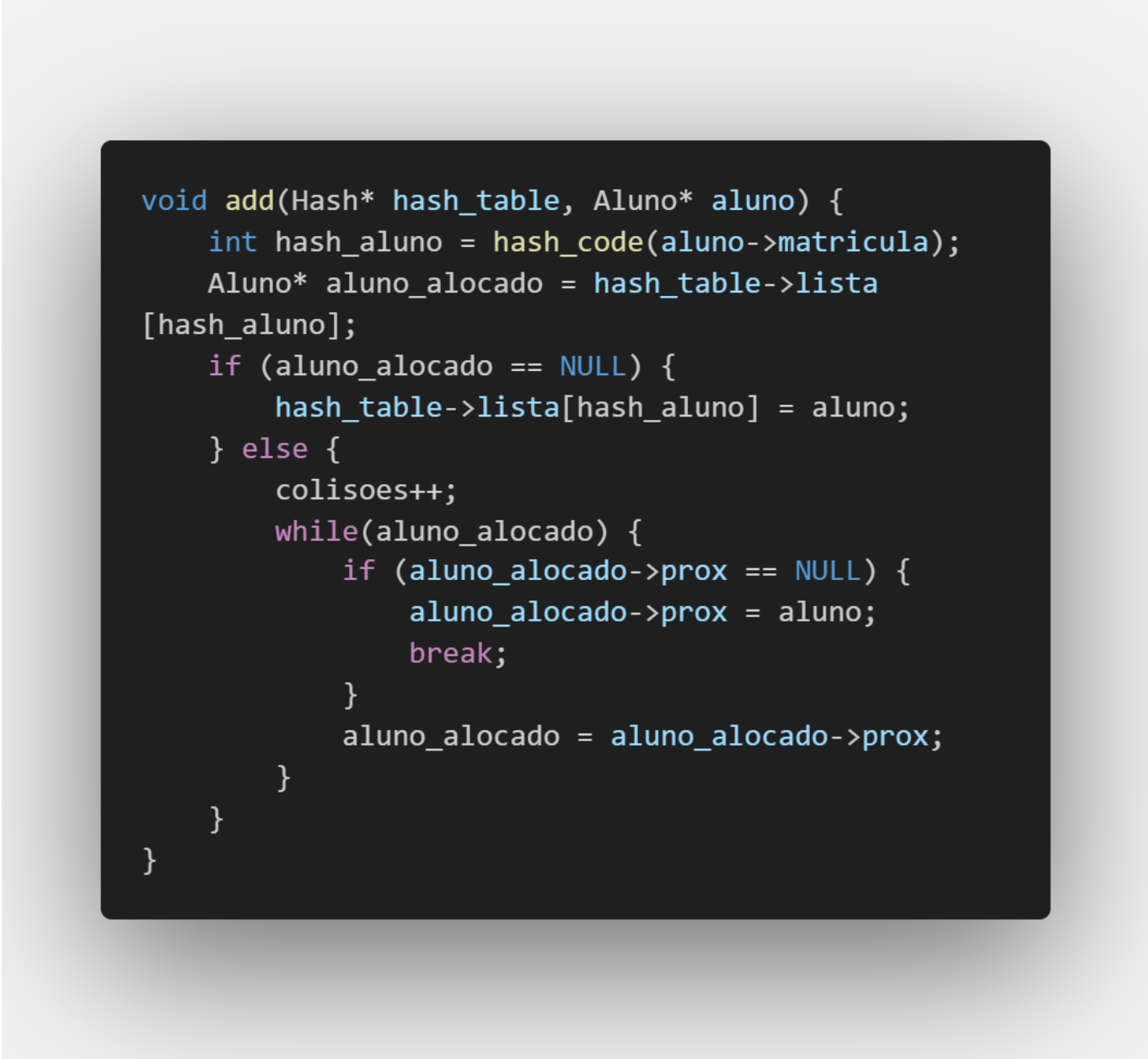
Temos a definição da estrutura e seus comportamentos no arquivo de cabeçalho. A criação se dá no método acima chamado "criar_tabela_hash", alocando espaço em memoria, faz a

verificação se houve erro e alimenta os espaços da tabela com nulo. O tamanho é definido pela variavel de pré processamento MAX_LISTA, que esta definida no documento de cabeçalho.



```
int hash_code(int matricula) {
    return (matricula & 0x7FFFFFFF) % MAX_LISTA;
}
```

Aqui temos a definição de como os identificadores da tabela hash serão gerados para cada elemento a ser inserido na estrutura.



```
void add(Hash* hash_table, Aluno* aluno) {
    int hash_aluno = hash_code(aluno->matricula);
    Aluno* aluno_alocado = hash_table->lista
[hash_aluno];
    if (aluno_alocado == NULL) {
        hash_table->lista[hash_aluno] = aluno;
    } else {
        colisoes++;
        while(aluno_alocado) {
            if (aluno_alocado->prox == NULL) {
                aluno_alocado->prox = aluno;
                break;
            }
            aluno_alocado = aluno_alocado->prox;
        }
    }
}
```

Esse comportamento aloca nossos dados em uma lista encadeada dinâmica. Primeiramente fazemos o calculo para criar o hash de identificação da estrutura, buscamos na tabela se há algum dado com o hash gerado. Não havendo fazemos a atribuição dos dados no indice correspondente. Mas se há dados para o mesmo indice, percorremos até o local em que deve ser feita a alocação dos novos dados.

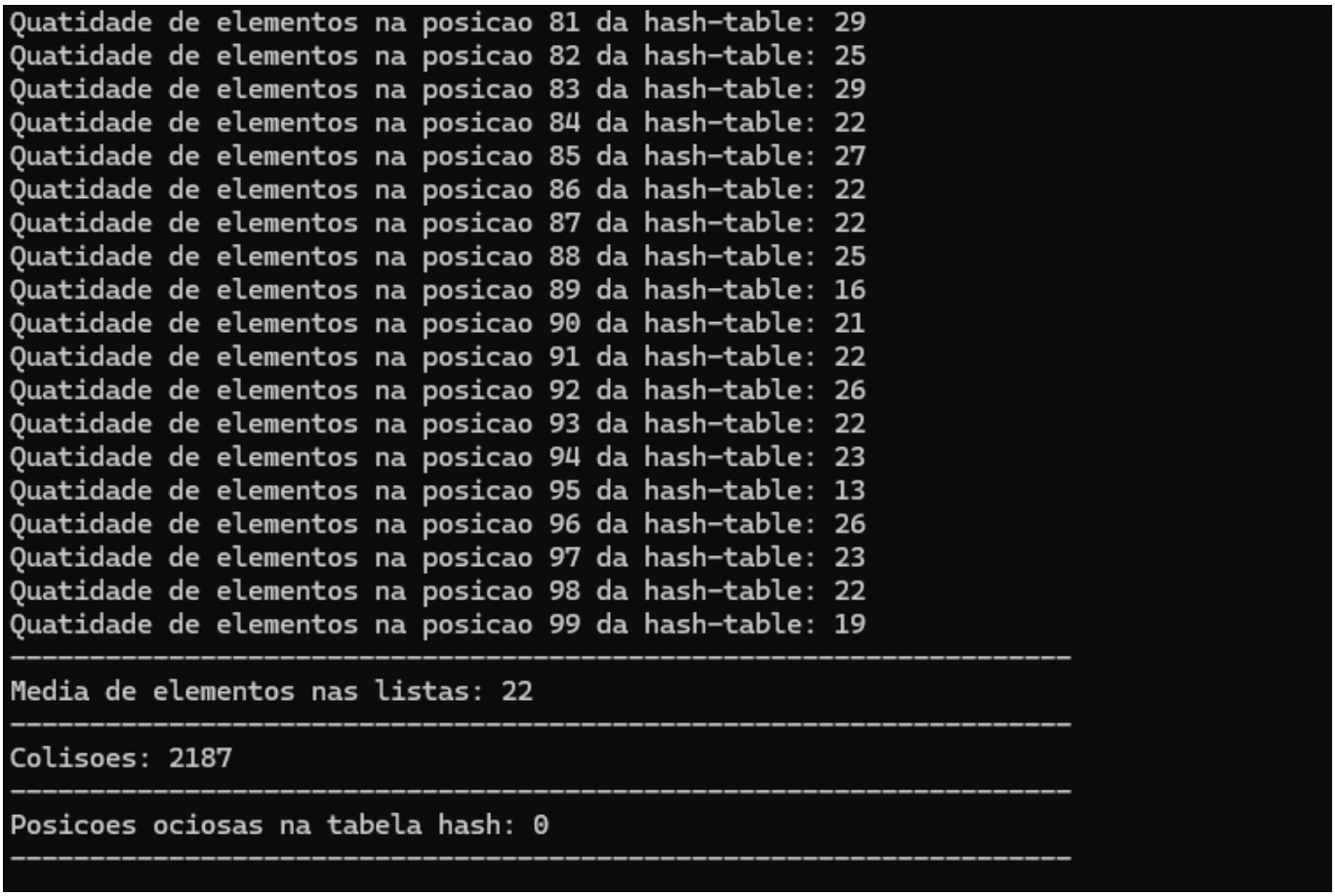
```c
int acumulador = 0;
int ociosa = 0;
for (int i = 0; i < MAX_LISTA; i++) {
    Aluno* aluno = hash->lista[i];
    if (aluno == NULL) {
        ociosa++;
    }
    int count = 0;
    while(aluno) {
        count++;
        aluno = aluno->prox;
    }
    printf("Quatidade de elementos na posicao %d da hash-table: %d\n", i, count);
    acumulador += count;
}
printf("-----------------------------------------------------------------------\n");
printf("Media de elementos nas listas: %d\n", acumulador/MAX_LISTA);
printf("-----------------------------------------------------------------------\n");
printf("Colisoes: %d\n", get_colisoes());
printf("-----------------------------------------------------------------------\n");
printf("Posicoes ociosas na tabela hash: %d\n", ociosa);
printf("-----------------------------------------------------------------------\n");
```

Aqui fazemos a leitura dos dados para fazer a aferição de algumas questões.

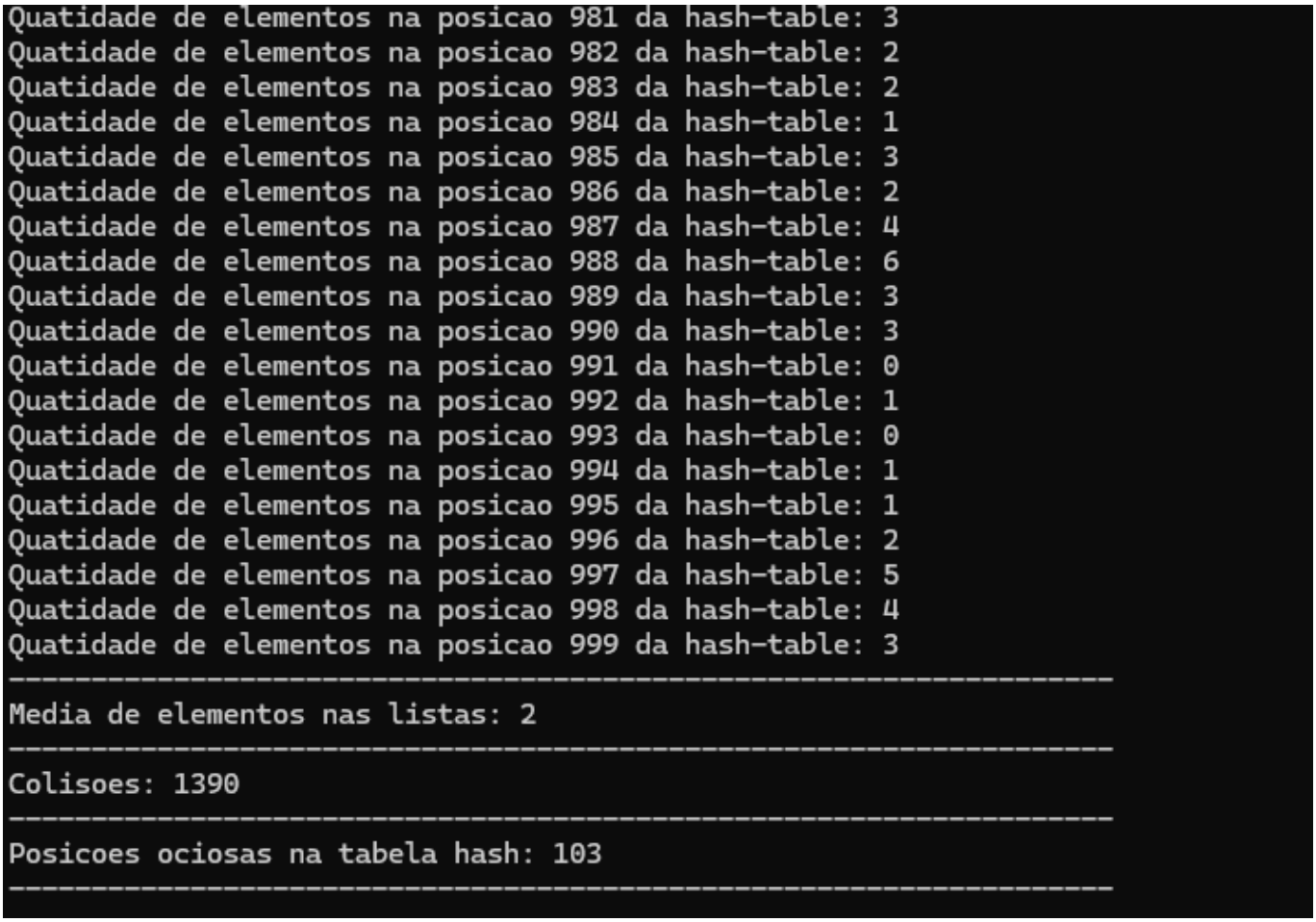
Resultados

```
Quatidade de elementos na posicao 0 da hash-table: 259
Quatidade de elementos na posicao 1 da hash-table: 222
Quatidade de elementos na posicao 2 da hash-table: 238
Quatidade de elementos na posicao 3 da hash-table: 216
Quatidade de elementos na posicao 4 da hash-table: 230
Quatidade de elementos na posicao 5 da hash-table: 231
Quatidade de elementos na posicao 6 da hash-table: 250
Quatidade de elementos na posicao 7 da hash-table: 203
Quatidade de elementos na posicao 8 da hash-table: 232
Quatidade de elementos na posicao 9 da hash-table: 206
-----------------------------------------------------------------------
Media de elementos nas listas: 228
-----------------------------------------------------------------------
Colisoes: 2277
-----------------------------------------------------------------------
Posicoes ociosas na tabela hash: 0
-----------------------------------------------------------------------
```

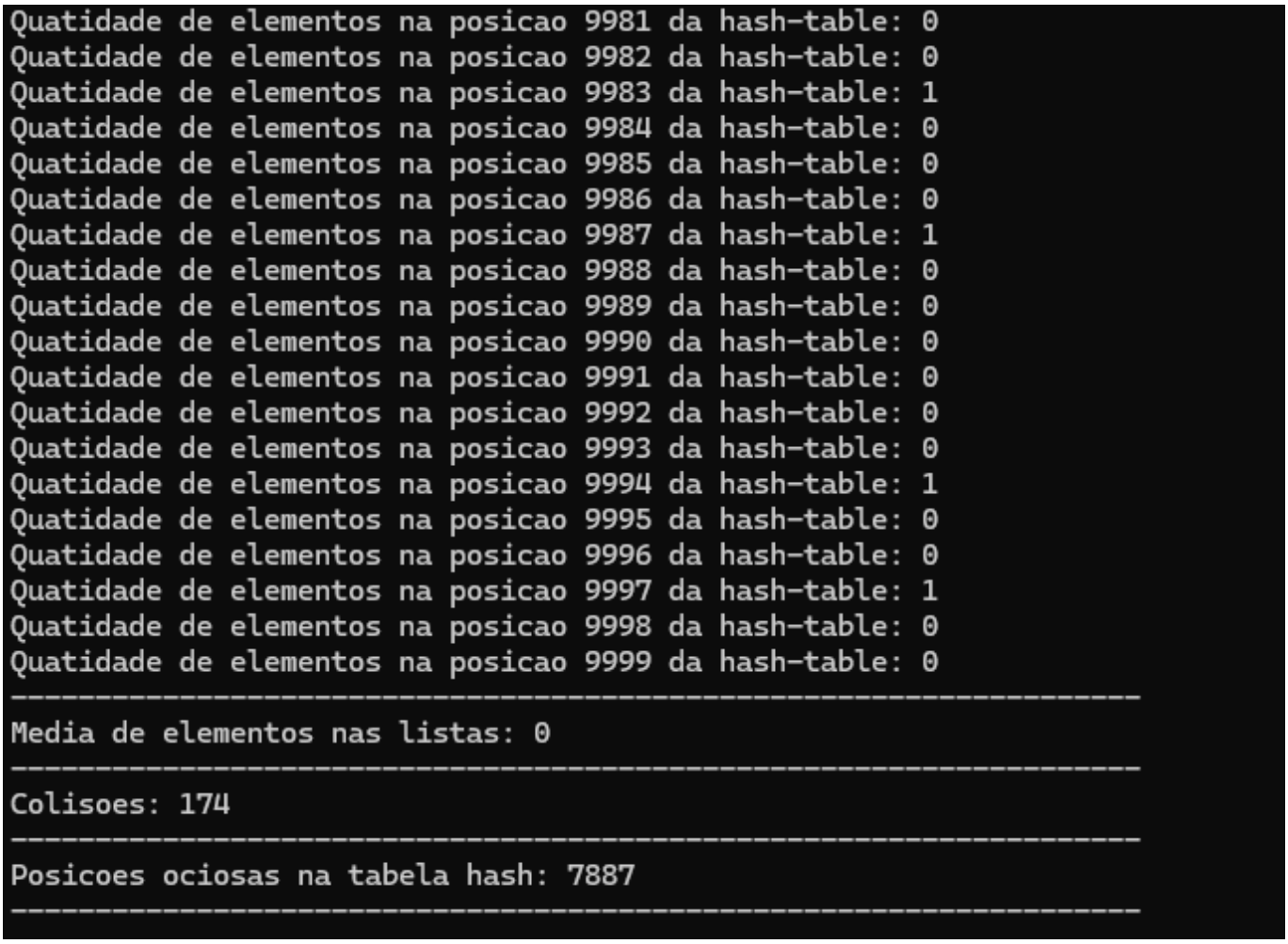
Resultado para uma estrutura com 10 posições na tabela Hash. Teremos 10 listas, a média de elementos de cada lista será 228 com 2277 colisões e sem posições ociosas.



Resultado para uma estrutura com 100 posições na tabela Hash. Teremos 100 listas, a média de elementos de cada lista será 22 com 2187 colisõese sem posições ociosas.



Resultado para uma estrutura com 1.000 posições na tabela Hash. Teremos 1.000 listas, a média de elementos de cada lista será 2 com 1390 colisões com 103 posições ociosas.



Resultado para uma estrutura com 10.000 posições na tabela Hash. Teremos 10.000 listas, a média de elementos de cada lista será 0 com 174 colisões com 7887 posições ociosas.
