# Estruturas de Dados

## Engenharia Informática

1º Ano - 2º Semestre

**Francisco Morgado**
Escola Superior de Tecnologia de Viseu

# 3. ESTRUTURAS DINÂMICAS

*3.1 Stacks (Pilhas)*

*3.2 Filas de espera*

*3.3 Listas ligadas ordenadas*

*3.4 Listas bi-ligadas ordenadas*

*3.5 Árvores binárias*

***3.6 Hashing***

# 3.6 Hashing

*Conceito - Estrutura*

*Aplicação*

# Conceito - Estrutura

A estrutura *hashing* consiste numa tabela com entradas em que cada entrada tem uma chave diferente, única e possível de determinar de forma simples.

Com esta estrutura é possível fazer uma pesquisa por dispersão (hashing search) ou pesquisa por tabela.

Tipo Aluno definido anteriormente

**Lista**

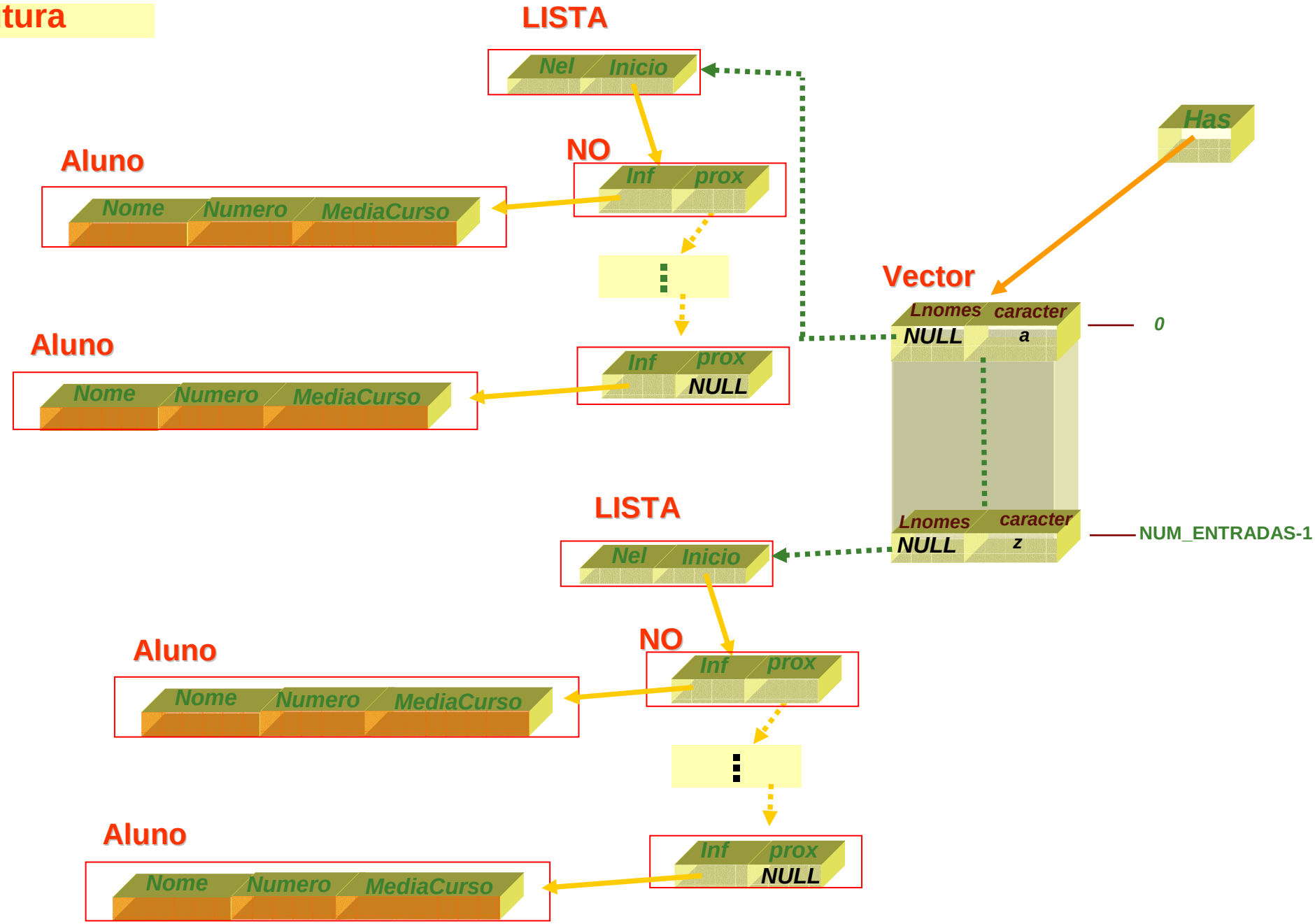
```
typedef struct no
{
    Aluno *inf;
    struct no *seg;
}No, *NO;
typedef struct
{
    int nElementos;
      // Número de Nos da Lista
    NO inicio;
      // Inicio da sequência dos Nos
}Lista, *LISTA;
```

Ordenação dada pelo 1º carácter do nome

***Hashing***

```
#define NUM_ENTRADAS 26
typedef struct
{
    char caracter;
    LISTA lnomes;
}NO_Hashing;
typedef struct
{
    NO_Hashing Vector[NUM_ENTRADAS];
}Hashing;
```

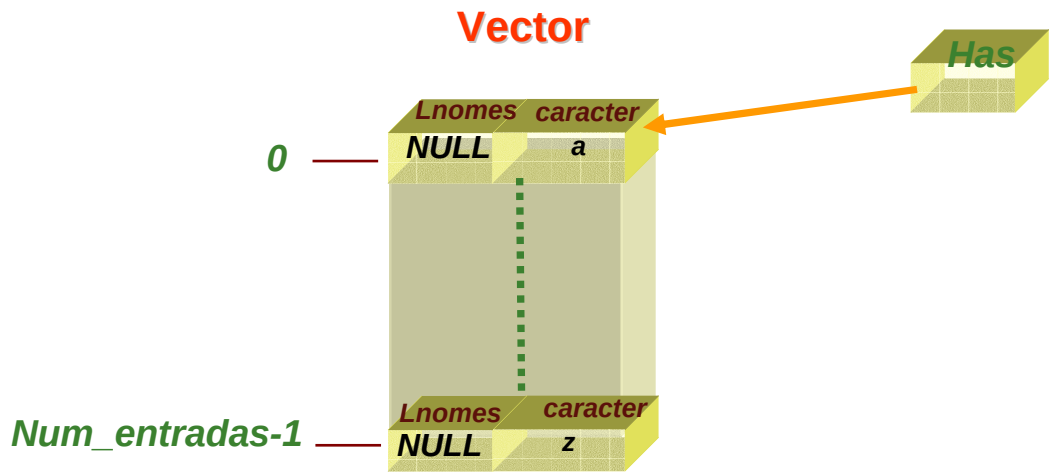
# Estrutura


LISTA
Nel
Inicio
NO
Inf
prox
Aluno
Nome
Numero
MediaCurso
Has
Vector
Lnomes
caracter
NULL
a
0
Inf
prox
NULL
Aluno
Nome
Numero
MediaCurso
LISTA
Nel
Inicio
Lnomes
caracter
NULL
z
NUM_ENTRADAS-1
NO
Inf
prox
Aluno
Nome
Numero
MediaCurso
Inf
prox
NULL
Aluno
Nome
Numero
MediaCurso

# Aplicação

## Criar um hashing

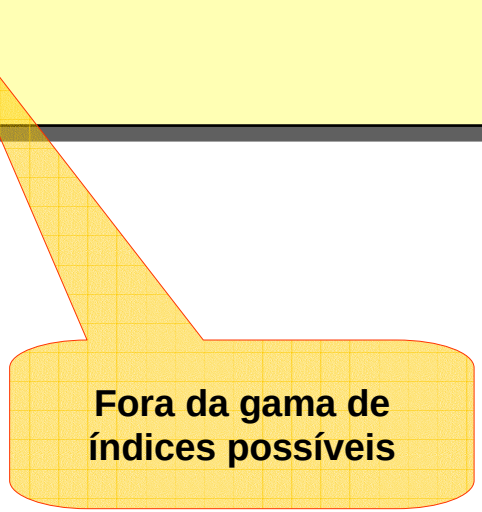
```
int main() {
 ...
 Hashing *alunos;
 alunos = criarHashing();
 ...
```




```
Hashing *criarHashing() {
  Hashing *Has = (Hashing *)malloc(sizeof(Hashing));
  char car = 'a';
  for ( int i = 0; i < NUM_ENTRADAS; i++) {
    Has->Vector[i].caracter = car++;
    Has->Vector[i].lnomes = NULL; // a lista ainda não é criada
  }
  return Has;
}
```

## Determinar a posição que um dado nome ocupa no hashing

```
int funcaoHashing(char *nome)
// Esta função permite determinar a posição no hashing,
// mediante o nome dado.
// Notar que a posição na estrutura depende do 1º caracter do nome
{ // Assume-se que nome!= NULL
      if (nome[0] >= 'A' && nome[0] <= 'Z')
         return nome[0] - 'A';
      else
            if (nome[0] >= 'a' && nome[0] <= 'z')
              return nome[0] - 'a';
      return -1;
}
```



## Inserir um aluno na estrutura definida

```
...
 switch(opcao) {
    case 1: // Inserir novo elemento
      no_criar = criarNo();
      lerNo(no_criar);
      inserirAlunoHashing(alunos, no_criar);
      break;
   ...
```

```
void inserirAlunoHashing(Hashing *Has, NO novo) {
  if (!Has) {
      printf("\nVector de Hashing não definido...");
      return;
  }
  // temos de determinar a posição onde se vai inserir
  int pos = funcaoHashing(novo->inf->nome);
  if ((pos >= 0) && (pos < NUM_ENTRADAS)) { // Será posição válida?
    if (!Has->Vector[pos].lnomes) // Se ainda não existe lista, cria-a
        Has->Vector[pos].lnomes = criarLista();
    inserirOrd(Has->Vector[pos].lnomes, novo);
  }
  else
    printf("ATENCAO: Erro na função <FuncaoHashing> \n");
}
```

## Mostrar toda a informação da Estrutura

```
...
case 4:  //  Mostrar
     mostrarHashing(alunos);
     break;
  ...
```

```
void mostrarHashing(Hashing *Has) {
 if (!Has) {
   printf("\nVector de Hashing não definido...");
   return;
 }
 char car = 'a';
 for ( int i = 0; i < NUM_ENTRADAS; i++,car++) {
   if (Has->Vector[i].lnomes) {
     printf("\nPosição [%d][%c], tem %d Alunos:",i, car,
                                 Has->Vector[i].lnomes->nElementos);
     listarLista(Has->Vector[i].lnomes);
   }
   else
     printf("\nPosição [%d][%c] tem 0 Alunos (ainda sem Lista)!",
                                                        i, car);
 }
}
```

Incrementadas 2 variáveis

## Pesquisar : verificar se um aluno está na estrutura

```
...
case 2: // Pesquisar um elemento no vector de hashing
    no_pesquisar = criarNo();
    printf("Introduza nome do aluno a pesquisar: ");
    gets(no_pesquisar->inf->nome);
    no_pesquisar = pesquisarHashingAluno(alunos,no_pesquisar);
    if (no_pesquisar) { // encontrado
      printf("Foi encontrado um elemento: ");
      mostrarNo(no_pesquisar); printf("\n");
    }
    else
      printf("Não existe nenhum elemento com esse nome...\n");
    break;
...
```

```
NO pesquisarHashingAluno(Hashing *Has, NO A) {
  if (!Has) return NULL;
  // Temos de determinar a posição onde se vai procurar
  int pos = funcaoHashing(A->inf->nome);
  if ((pos >= 0) && (pos < NUM_ENTRADAS))   // Se posição válida...
    return pertenceIterativo(Has->Vector[pos].lnomes, A);
  else
     printf("\nOcorreu um erro ao calcular a posicao de hashing!");
  return NULL;
}
```

## Pesquisar : verificar se um aluno está na estrutura

```
...
case 3: // Pesquisar um elemento no vector de hashing, por número
        // (que não é o critério de ordenação)
  printf("Introduza número do aluno a encontrar: ");
  scanf("%d", &nal);
  no_pesquisar = pesquisarHashingAlunoNumero(alunos, nal);
  if (no_pesquisar) { // encontrado
    mostrarNo(no_pesquisar); printf("\n");
  else
    printf("Não foi encontrado nenhum aluno com o número %d!\n",nal);
  break;
...
```

```
NO pesquisarHashingAlunoNumero(Hashing *Has, int nal) {
// Não se pode tirar partido da ordenação...
  if (!Has)
    return NULL;
  NO p;
  for ( int i = 0; i < NUM_ENTRADAS; i++) {
    p = pertenceIterativoNumero(Has->Vector[i].lnomes, nal);
    if (p)
      return p;
  }
  return NULL;
}
```

Se já encontramos o aluno com o número pretendido numa das 26 listas, não é necessário ir até ao fim do Hashing

## Retirar um aluno da estrutura

```
...
case 7: // Retirar um elemento do vector de hashing
  no_remover = criarNo();
  printf("Introduza nome do aluno a eliminar: ");
  gets(no_remover->inf->nome);
  no_remover = RemoverHashingNo(alunos, no_remover);
  if (no_remover) { // encontrado
    mostrarNo(no_remover);
    libertarNo(no_remover);
  }
  else printf(" Não existe nenhum elemento com esse nome!\n");
  break;
...
```

```
NO RemoverHashingNo(Hashing *Has, NO A)
{
  if (!Has)
    return NULL;
  // Temos de determinar a posição onde se vai remover
  int pos = funcaoHashing(A->inf->nome);
  if ((pos >= 0) && (pos < NUM_ENTRADAS)) // posição válida?
    return removerNo(Has->Vector[pos].lnomes, A);
  return NULL;
}
```

## Retirar um aluno da estrutura

```
...
case 8: // Retirar um elemento do vector de hashing, por número
        // (que não é o critério de ordenação)
  printf("Introduza número do aluno a eliminar: ");
  scanf("%d", &nal);
  no_remover = removerHashingNoNumero(alunos, nal);
  if (no_remover) { // encontrado
      mostrarNo(no_remover);
      libertarNo(no_remover);
  }
  else printf(" Não existe nenhum elemento com o nº %d!\n",nal);
  break;
...
```

```
NO removerHashingNoNumero(Hashing *Has, int nal) {
// Não se pode tirar partido da ordenação...
  if (!Has)
    return NULL;
  NO p;
  for ( int i = 0; i < NUM_ENTRADAS; i++) {
    p = removerNoNumero(Has->Vector[i].lnomes, nal);
    if (p)
      return p;
  }
  return NULL;
}
```

Se removemos o aluno (numa das 26 listas), não é necessário ir até ao fim do Hashing

## Contar o número de alunos que se encontram na Estrutura

```
...
case 5: // Mostra número total de alunos
  printf("\n No total, existem %d alunos\n",contarAlunosHashing(alunos));
  break;
...
```

```
int contarAlunosHashing(Hashing *Has)
{
    if (!Has) return 0;
    int cont = 0;
    for (int i = 0; i < NUM_ENTRADAS; i++)
       if (Has->Vector[i].lnomes)
          cont += Has->Vector[i].lnomes->nElementos;
    return cont;
}
```

## Determinar qual a letra inicial (do nome) que ocorre mais vezes

```
...
case 6: // Letra inicial mais frequente
  printf("\n A letra inicial mais frequente nos ",
  printf("nomes é o '%c'\n", detCaracterMaisVezes(alunos));
  break;
...
```

```
char detCaracterMaisVezes(Hashing *Has) {
 // Basicamente, este problema consiste em percorrer o vector do Hashing
 // e determinar qual a posição onde a lista de alunos tem mais elementos
    if (!Has) return '?';
    char car = '?';
    int nvezes = -1;
    for (int i = 0; i < NUM_ENTRADAS; i++)
        if (Has->Vector[i].lnomes)
            if (Has->Vector[i].lnomes->nElementos > nvezes)
            {
                nvezes = Has->Vector[i].lnomes->nElementos;
                car = Has->Vector[i].caracter;
            }
    return car;
}
```

## Destruir toda a estrutura

```
...
case 9: //  Limpar vector  Hash
    destruirHashing(alunos);
    alunos = NULL ; break;
...
```

```
void destruirHashing(Hashing *Has) {
  if (!Has)    return;
  for (int i = 0; i < NUM_ENTRADAS; i++) {
    destruirLista(Has->Vector[i].lnomes); //destrói nós da lista
      if (Has->Vector[i].lnomes) { //para destruir a própria lista
        free(Has->Vector[i].lnomes);
        Has->Vector[i].lnomes = NULL; // desnecessário pq depois fazemos free do vector
      }
  }
  free (Has); // liberta vector
}
```

destrói nós da lista, mas não a própria lista...

```
void destruirLista(LISTA L) { // remove e liberta todos os elementos da lista
  if (!L)  return;
  // Destroi a Lista e toda a informação que ela contêm
  NO p = L->inicio;
  while (p) {
    NO ant = p;
    p = p->seg;
    libertarNo(ant);
  }     // no fim do ciclo todos os nós da lista estão 'destruídos'
  L->inicio = NULL;
  L->nElementos = 0; // desnecessário pq depois fazemos free da lista...
}
```