Programação I

Listas ligadas simples

Estrela Ferreira Cruz

1

Objetivos da aula

Objetivos da aula:

Listas ligadas (ou encadeadas) simples:

- Apresentação do conceito;
- Definição da estrutura de dados que representa uma lista ligada.
- Vantagens de desvantagens do uso de listas ligadas em relação a outras estruturas de dados.
- Operações de manuseamento de listas: inserção (início, meio e fim), remoção (início, meio e fim), percorrer a lista, etc.
- Apresentação de exemplos práticos.

2

Listas ligadas

Alocação de memória (recapitulação)

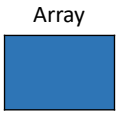
- Vetor estático – a dimensão é estática, e tem de ser definida no momento da compilação
- Alocação dinâmica de um vetor – a memória é alocada quando o programa se encontra em execução (malloc() ou calloc()), sendo possível alterar o seu tamanho (realloc). No entanto este vetor vai ocupar um espaço contíguo na memória, que pode ser difícil de conseguir.
- Alocação dinâmica de uma estrutura ligada – cada elemento é alocado individualmente à medida que é necessário, com grande flexibilidade. No entanto, exige mais espaço para as ligações entre si (espaço ocupado pelo apontador para o seguinte), e o acesso aos elementos é mais demorado se compararmos com o caso do vetor. A alocação de novos elementos na lista pode ser mais rápido que vetores dinâmico (quando se usa o realloc()).

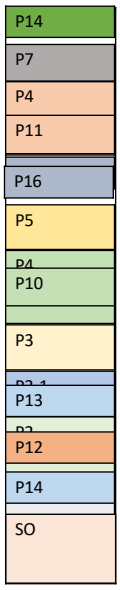
3

Listas ligadas

Alocação de memória (recapitulação)

Onde reservar um espaço para este bloco (array)?





4

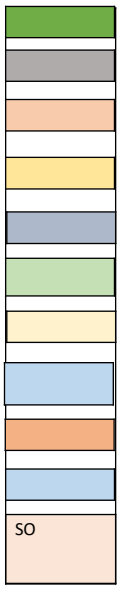
Listas ligadas

Alocação de memória (recapitulação)

Onde reservar um espaço para este bloco (array)?

Array

Memoria



EstrelaFCruz 5

5

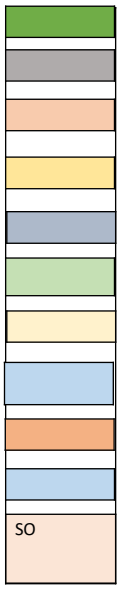
Listas ligadas

Alocação de memória (recapitulação)

Onde reservar um espaço para este bloco (array)?

Array

Memoria



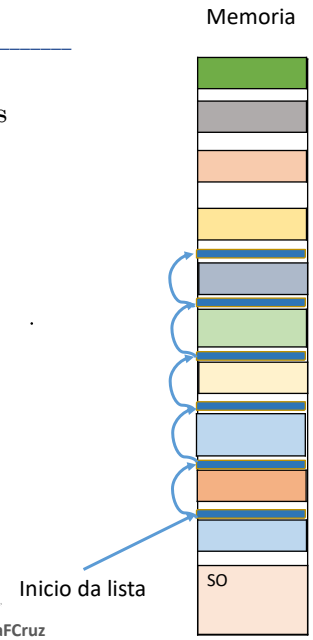
EstrelaFCruz 6

6

Listas ligadas

Como saber onde ficaram os nossos registos?

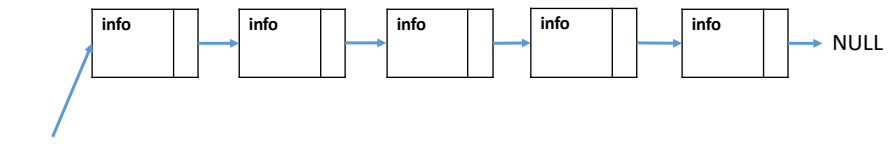
Temos assim
lista encadeadas ou
lista ligadas
(linked list)



7

Listas ligadas

Uma lista ligada pode ser representada da seguinte forma:



Cada elemento da lista contem:
- Informação (que se pretende armazenar)
- O endereço de memória do próximo da lista (é um apontador)

8

Listas ligadas

Listas ligadas

- É uma estrutura de dados constituída por um conjunto de elementos conhecidos como nós, ligados entre si.
- Cada elemento (ou nó) contém um campo que indica a localização do elemento seguinte na lista, ou seja, contém uma referência para próximo elemento de forma a que todos os nós ligados formem uma lista ligada.
- O primeiro nó da lista fica referenciado por uma variável do mesmo tipo do nó chamada "header" (ou cabeça).
- Para chegar a um dos elementos temos de passar por outros elementos, exceto no caso do primeiro (e do último se a lista também tiver uma ligação direta à sua cauda)

9

Listas ligadas

Vantagens do uso listas ligadas

- A grande vantagem é que não necessita de um espaço contíguo de memória.
- A memória pode ser reservada e libertada conforme vai sendo necessário durante o programa. Isto porque as listas ligadas são implementadas usando variáveis dinâmicas.
- Permitem usar o espaço de memória necessário sem precisar de reservar espaço para o pior cenário.
- Grande rapidez nas operações de remoção e inserção de registos quando comparado com vetores dinâmicos.

10

Listas ligadas

Operações mais usuais sobre as listas:

Tendo em conta que os elementos podem ser inseridos, ou removidos em qualquer posição na lista, teremos as seguintes operações sobre listas:

- imprimir os dados dos elementos da lista para o ecrã
- inserção/remoção no inicio da lista
- inserção/remoção no meio da lista
- inserção/remoção no fim da lista
- calcular o comprimento da lista
- destruição da lista (libertar a memória)

11

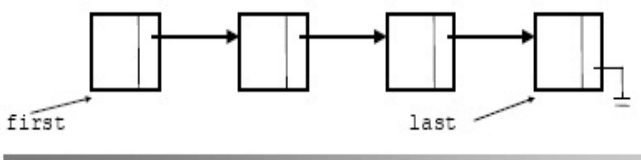
Listas ligadas

Listas Ligadas

A representação em C de listas ligadas é conseguida utilizando estruturas com duas partes:

- a primeira parte contém a informação a armazenar na lista.
- a segundo é um apontador para o elemento seguinte na lista.

A representação gráfica pode ser a seguinte:



12

Listas ligadas

Implementação de listas ligadas em C

- Define-se uma estrutura que representa a informação a armazenar na lista e um apontador para o próximo elemento da lista.

No exemplo seguinte temos uma lista com a informação sobre pessoas.

```
typedef struct aluno{
        char nome[100];
        int numero;
        char morada [200];
}INFO;
typedef struct Elem {
        INFO node;              //conteúdo do nó.
        struct Elem *next;      //apontador para o próximo nó
}ELEMENTO;
```


Listas ligadas

Implementação de listas ligadas em C

- Em seguida declaramos um apontador para o inicio da lista, ou seja, este apontador deve conter o endereço de memória do primeiro elemento da lista. O apontador pode ser declarado da seguinte forma:

  ELEMENTO *inilista=NULL;

Notação: Se aux é uma célula da lista então aux.node é o conteúdo da célula e aux.next é o endereço de memória da próxima célula

- Se aux é o endereço de memória de uma célula (ELEMENTO *aux=NULL;) , então aux->node é o conteúdo da célula e aux->next é o endereço de memória da próxima célula da lista.
- Se aux aponta para a última célula da lista então aux->next é NULL.

Podemos representar "(*aux).next" por "aux->next" e "(*aux).node" por "aux->node".

Listas ligadas

O último elemento da lista deve ter o seu campo next a NULL.

Para percorrer toda a lista basta avançar célula a célula, começando pela primeira e terminado quando o apontador para o seguinte for NULL, como se mostra em baixo.

```
ELEMENTO *ptr=NULL;
for (ptr=iniLista; ptr != NULL; ptr=ptr->next){
     // ….
}
// ou usar um ciclo while
ptr = iniLista;
while (ptr!=NULL) {
      // …
      ptr=ptr->next;
}
```


15

Listas ligadas

Para imprimir para o ecrã o conteúdo da lista de alunos, podemos usar a seguinte função:

```
void imprimeLista(ELEMENTO *iniLista) {
ELEMENTO *aux=NULL;
if (iniLista==NULL){
      printf("lista vazia\n");
      return;
}
For (aux=iniLista;aux!=NULL;aux=aux->next){
      printf ("%d - %s", aux->node.numero,
                        aux->node.nome);
}
}
```


16

Listas ligadas

Para imprimir para o ecrã o conteúdo da lista de alunos, podemos usar a seguinte função:

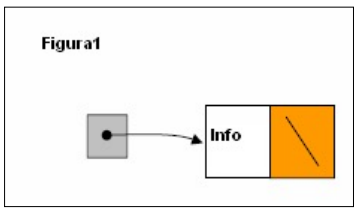
```
void imprimeElementosDaLista(ELEMENTO *iniLista) {
ELEMENTO *aux=NULL;
if (iniLista==NULL) {printf ("lista vazia\n");}
else {
      aux = iniLista;
      while (aux!=NULL){
            printf ("%d - %s", aux->node.numero,
                        aux->node.nome);
            aux = aux->next;
      }
}
}
```

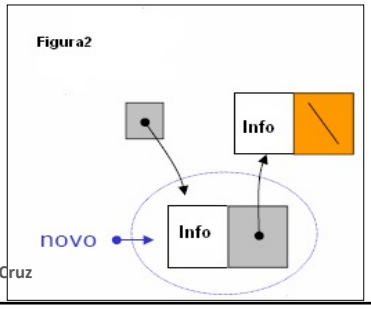

17

Listas ligadas

Inserção Inicio

- Em primeiro lugar é necessário reservas espaço memória para o novo elemento;
- depois é necessário colocar o apontador para o elemento seguinte da lista, do novo elemento, apontar para o inicio da lista (Figura2), se a lista não está vazia.
- o apontador para o seguinte, fica a NULL, se estivermos a inserir numa lista vazia (figura1).
- o apontador para a cabeça da lista passa a apontar para o novo elemento.

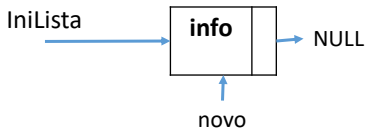




18

Listas ligadas

Inserir o primeiro elemento na lista

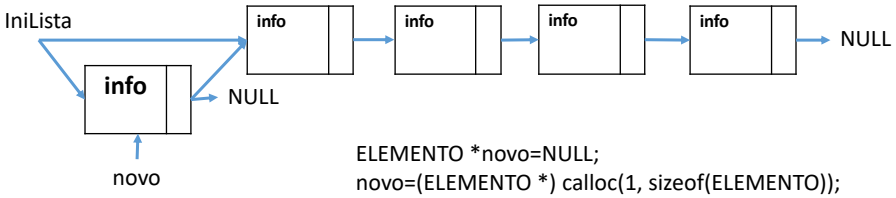



```
ELEMENTO *novo=NULL;
novo=(ELEMENTO *) calloc(1, sizeof(ELEMENTO));
novo->info=new_info;
novo->seguinte=NULL
iniLista=novo;
```


19

Listas ligadas

Inserir um elemento no inicio da lista que já contem outros elementos




```
ELEMENTO *novo=NULL;
novo=(ELEMENTO *) calloc(1, sizeof(ELEMENTO));
novo->info=new_info;
novo->seguinte=NULL
novo->seguinte=iniLista;
iniLista=novo;
```


20

Listas Ligadas

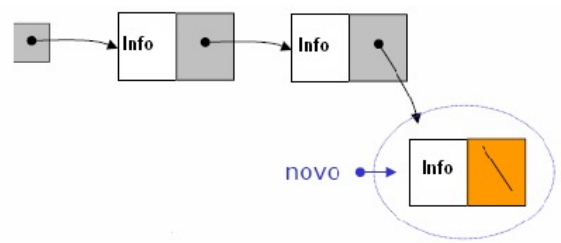
Inserção de um elemento no início da lista

```
int inserIniLista(ELEMENTO **iniLista, INFO new_info){
 ELEMENTO *novo=NULL;
  novo=(ELEMENTO *)calloc(1,sizeof(ELEMENTO));
  if (novo==NULL){
     printf("Erro ao reservar memória\n"); return -1;
  }
  novo->node = new_info;
  novo->next = NULL;
  if (*iniLista==NULL){*iniLista = novo;}
  else {
      novo->seguinte=*inilista;
      *iniLista=novo;
  }
  return 0;
}
```


Listas ligadas

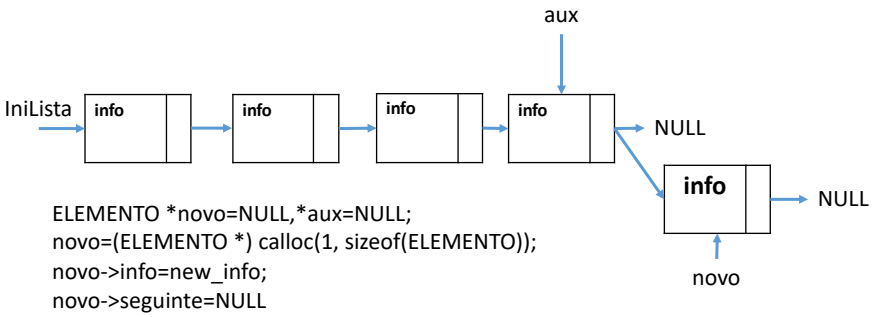
Inserção no fim

- Em primeiro lugar é necessário reservar memória para o novo elemento.
- Percorre-se a lista até chegar ao último elemento da lista.
- Coloca-se o elemento seguinte da lista (do último), apontar para o novo elemento da lista.
- O apontador para o seguinte, do novo elemento tem que ficar apontar para NULL .



Listas ligadas

Inserir no fim da lista:




```
ELEMENTO *novo=NULL,*aux=NULL;
novo=(ELEMENTO *) calloc(1, sizeof(ELEMENTO));
novo->info=new_info;
novo->seguinte=NULL
aux=iniLista;
While (aux->seguinte!=NULL){
        aux=aux->seguinte;
}
aux->seguinte=novo;
```


23

Listas Ligadas

Função que insere no fim da lista:

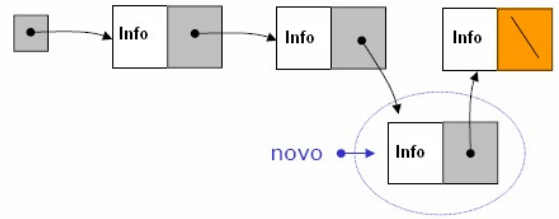
```
int inserFimLista (ELEMENTO **iniLista, INFO newElem){
   ELEMENTO *aux=NULL, *novo=NULL;
   novo = (ELEMENTO *) malloc(sizeof(ELEMENTO));
   if (novo==NULL) {printf("out of memory\n");return -1;}
   novo->node = newElem;
   novo->next = NULL;
   if (*iniLista == NULL) {    //lista vazia - vai inserir 1º elem
      *iniLista = novo;
   }
   else {
      aux = *iniLista;
      while (aux->next != NULL){ //percorre a lista até ao fim
          aux = aux->next;
      }
      aux->next = novo;      //acrescenta o novo elemento
   }
   return 0;
}
```


24

Listas ligadas

Inserção no meio

- Em primeiro lugar é necessário reservas memória para o novo elemento.
- Depois colocar o apontador para o seguinte, do novo elemento apontar para o elemento que vai ficar na posição seguinte da lista.
- Por fim, o apontador para o seguinte, do elemento que fica na posição anterior, apontar para o novo elemento.



25

Listas Ligadas

Função que insere no meio da lista:

- reservar espaço para novo elemento
- verificar a posição onde inserir
- guardar o elemento anterior
- colocar o next do novo elemento apontar para o seguinte
- colocar o next, do elemento anterior ao que vai ser inserido a apontar para o novo elemento.

```
void inserirMeio(ELEMENTO **iniLista, INFO New_elem){
  ELEMENTO *ap_novo=NULL, *anterior=NULL;
   …..
   ap_novo->next=anterior->next;
   anterior->next=ap_novo;
   …..
};
```


26

Listas ligadas

- A inserção no meio é normalmente usada numa inserção ordenada por um determinado campo.
- Para isso é necessário selecionar em que posição o novo elemento vai ser acrescentado: no início, meio ou fim.
- No algoritmo apresentado na página seguinte faz-se a inserção num alistas ordenada pelo campo numero.

27

Listas Lig

Inserção numa lista ordenada pelo numero.

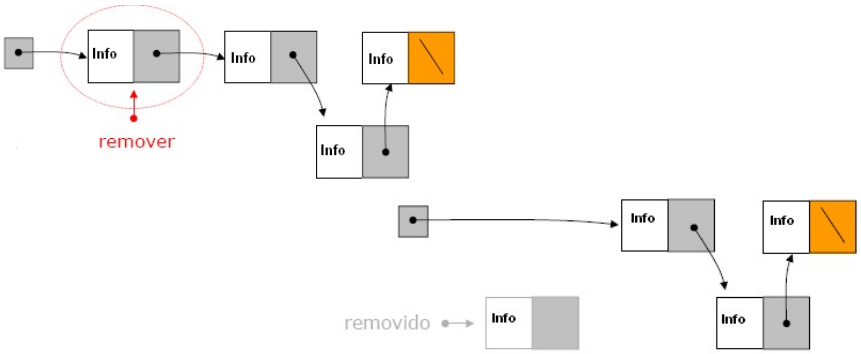
```
int inserirOrdenado(INFO new_nodo, ELEMENTO **iniLista) {
ELEMENTO *aux=*iniLista, *novo, *anterior=NULL;
novo = (ELEMENTO  *) calloc(1, sizeof(ELEMENTO));
If (novo==NULL) {printf("Out of memory\n"); return -1;}
novo->nodo = new_nodo;
novo->next = NULL;
if (*iniLista == NULL){*iniLista = novo; return 0; } // inserir na lista vazia
aux=*iniLista;
while(aux!=NULL && aux->nodo.numero < new_nodo.numero) {
        anterior = aux;
        aux = aux->next;
}
if (anterior==NULL) {       // inserir no inicio
        novo->next=*iniLista;
        *iniLista=novo;
}
else {
        if (anterior->next==NULL) { // inserir no fim
                anterior->next=novo;
        }
        else {        // inserir no meio
                novo->next=anterior->next;
                anterior->next=novo;
        }
}
 return 0; }
```


28

Listas ligadas

Remoção do primeiro elemento da lista (remoção do inicio).

- Primeiro, colocar o apontador para o inicio da lista apontar para o elemento seguinte ao que vai ser removido.
- Libertar a memória ocupada pelo elemento a remover.

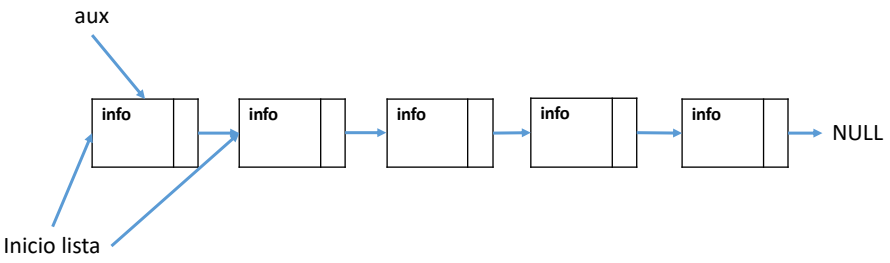


EstrelaFCruz

29

Listas ligadas

Remoção do 1ª elemento da lista, representada da seguinte forma:



EstrelaFCruz 30

30

Listas Ligadas

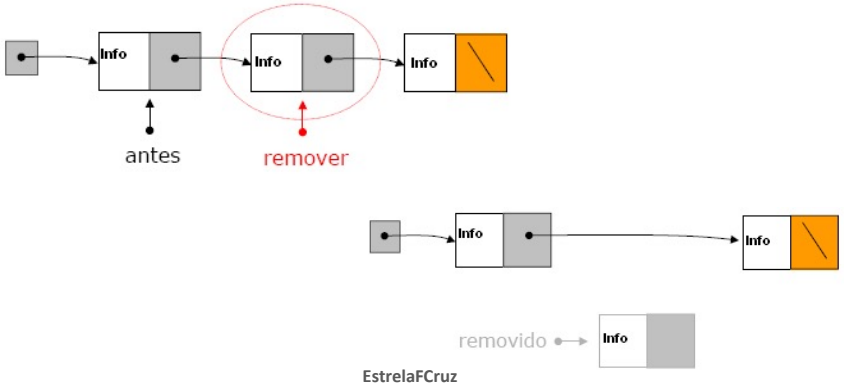
Remoção do primeiro elemento da lista.

```
int removeInicioLista(ELEMENTO **iniLista){
ELEMENTO *aux=NULL;
if (*iniLista == NULL) {
       printf ("Lista vazia");
       return -1;
}
aux=*iniLista;
*iniLista=aux->next;
free(aux);
return 0;
}
```


Listas ligadas

Remoção de um elemento do Meio

- Em primeiro lugar colocar um apontador (auxiliar) apontar para o elemento anterior ao que se pretende remover.
- Coloca-se o seguinte do anterior apontar para o elemento seguinte ao que se pretende remover.



Listas Ligadas

Função que remove do meio da lista deve:

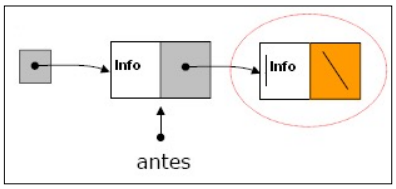
- identificar o elemento que vai ser removido
- guardar o elemento anterior ao que vai ser removido numa variável auxiliar
- colocar o *seguinte (next)*, do elemento anterior ao que vai ser removido a apontar para o elemento seguinte ao que vai ser removido.

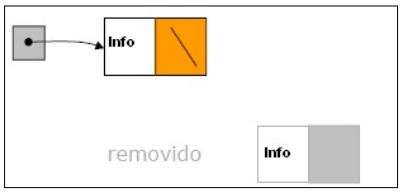
```
void removeMeioLista(ELEMENTO **iniLista,Info Elem){
  ELEMENTO *anterior=NULL, *actual=*iniLista;
      …..
  anterior->next = actual->next;
  free(actual);
       …..
};
```


Listas ligadas

Remoção do último elemento da lista (do Fim):

- Em primeiro lugar, colocar um apontador (auxiliar) apontar para o penúltimo elemento da lista
- Colocar o *seguinte* do penúltimo apontar para NULL
- Depois, libertar o espaço ocupado.





Listas Ligadas

Função que remove o elemento do fim da lista

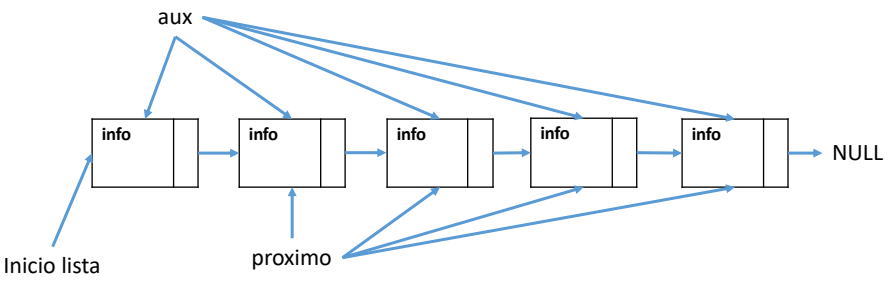
```
int removeFimLista(ELEMENTO **iniLista ){
ELEMENTO *aux=NULL, *ant=NULL;
if (*iniLista == NULL){
    printf("lista vazia");
    return;
}
aux = *iniLista;
if(aux->seguinte==NULL){//remover único elem. da lista
    *iniLista=NULL;
}
else{
    while(aux->next!=NULL){
       ant=aux;
        aux=aux->next;
    }
    ant->next=NULL; //remove o elem. do fim da lista
}
free(aux);
return 0;
}
```

EstrelaFCruz

35

Listas ligadas

Libertar memoria da lista, elemento a elemento:



EstrelaFCruz 36

36

Listas Ligadas

Devemos sempre assegurar-nos que não fica memória por libertar.

Para isso podemos implementar uma função que liberta toda a memória ocupada pela lista.

A função pode ser invocada quando a informação armazenada na lista deixa de ser necessária, como por exemplo no final do programa.

```
void libertaLista(ELEMENTO **iniLista){
ELEMENTO *aux=NULL;
ELEMENTO *proximo=NULL;
aux=*iniLista;
*iniLista=NULL;
while(aux!=NULL){
        proximo = aux->seguinte;
        free(aux);
        aux=proximo;
}
}
```


37

Listas Ligadas

Funções que calculam o comprimento da lista, ou seja, contam o numero de elementos da lista recebida como parâmetro:

getSizeIt() - recorre ao uso de iteratividade

getSizeRec()- recorre ao uso da recursividade.

```
int getSizeIt(ELEMENTO *iniLista){
ELEMENTO *aux1=NULL;
int totElem=0
aux1 = iniLista;
while(aux1!= NULL){
        totElem++;
        aux1 = aux1->next;
}
return totElem;
}
```

```
int getSizeRec(ELEMENTO *ap_lista) {
  if (ap_lista==NULL) { return 0;}
  return (1 + getSizeRec(ap_lista->next));
}
```


38

Listas Ligadas

Exercício:

1 – Implemente uma lista ligada que permita fazer a gestão (inserir/remover/listar e alterar) das inscrição para a meia maratona de Viana do Castelo. Sobre cada inscrição devemos armazenar o **nome**, **nCC, telefone** e **idade** do participante.

Bibliografia

- Programação Avançada Usando C, António Manuel Adrego da Rocha, ISBN: 978-978-722-546-0.
- Schildt, Herbert: C the complete Reference, McGraw-Hill, 1998.
- Algoritmia e Estruturas de Dados, José Braga de Vasconcelos, João Vidal de Carvalho, ISBN: 989-615-012-5.
- Linguagem C, Luís Manuel Dias Damas, ISBN: 972-722-156-4.
- Elementos de Programação com C - Pedro João Valente D. Guerreiro, 3ª edição, ISBN: 972-722-510-1.
- Introdução à Programação Usando C, António Manuel Adrego da Rocha, ISBN: 972-722-524-1.