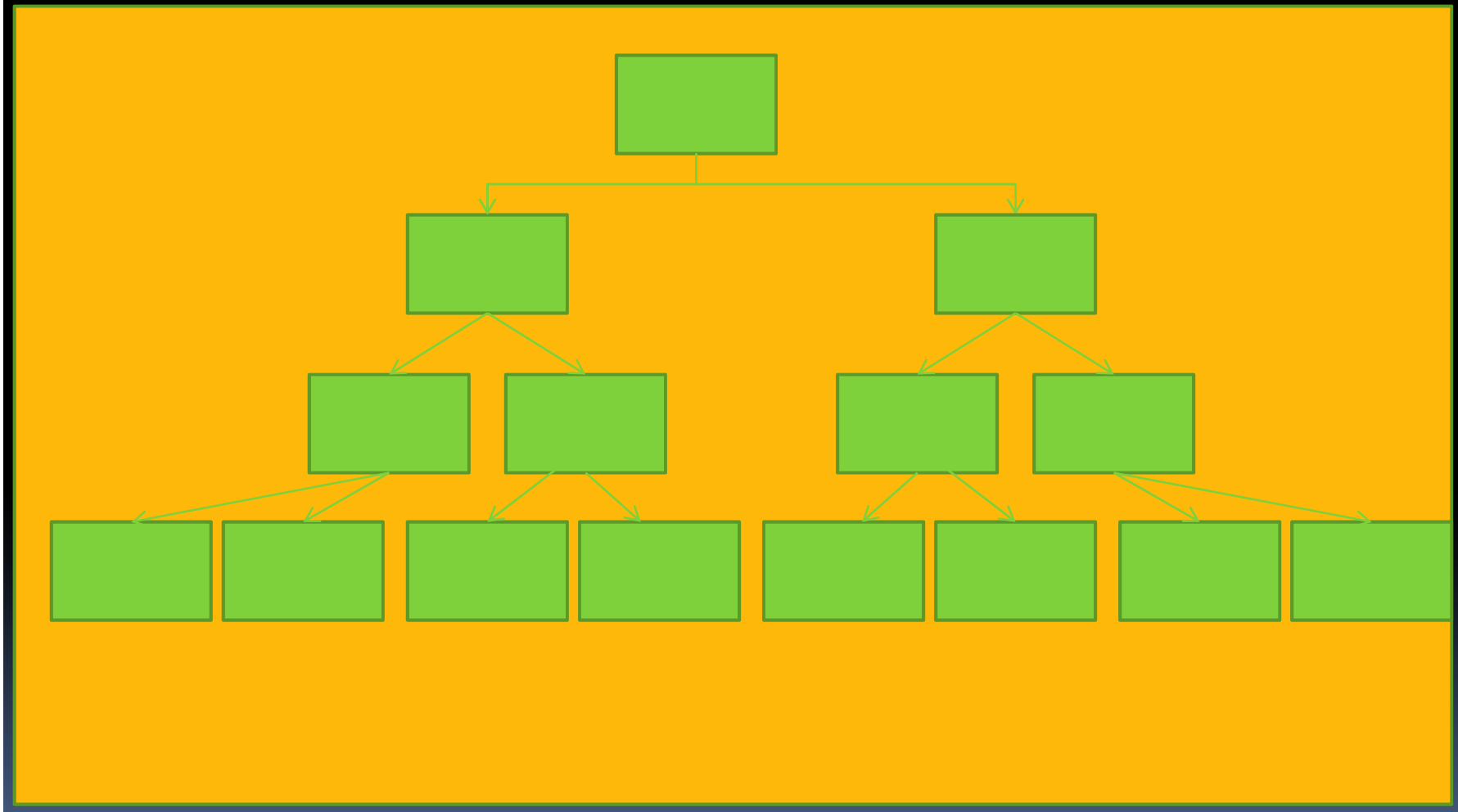
ESTRUTURAS DE DADOS EM LINGUAGEM C

Profa. Fernanda Argoud

Fev., 2010

Tipos de Dados



Tipos de Dados

- Primitivos – inteiro, real, lógico, caracter, ponteiro.
- Estrutura de Dados – modo específico de armazenamento e organização de dados na memória do computador.
- Implementa *grafos abstratos* na memória.

Tipos de Dados

- Um elemento de estrutura é chamado de NÓ.
- Os tipos de nós podem ser também classificados como :
- Homogêneos – todos os nós da estrutura são de um mesmo tipo. Ex: vetores e matrizes.
- Heterogêneos – os nós são compostos de campos de tipos diferentes. Ex: registros.

Tipos de Estruturas de Dados

- Estrutura de Dados estática – sua estrutura permanece a mesma (número de nós e posição dos mesmos, na memória). Ex: vetores.
- Estrutura de Dados dinâmica – sofrem alterações estruturais (inserção, remoção e variação do número de nós), à medida que são manipulados.

Operações sobre Dados

- CRIAÇÃO – momento de alocação de memória para o nó da estrutura de dados;
- PERCURSO – acesso a todos os nós da estrutura, ao mesmo tempo;
- BUSCA – por um nó específico na estrutura;
- ALTERAÇÃO – no conteúdo de um nó específico da estrutura;

Operações sobre Dados

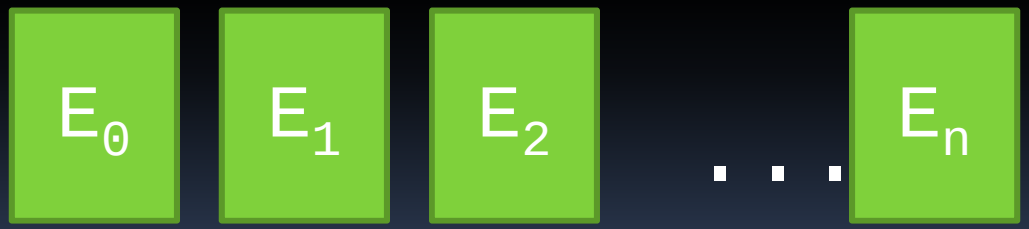
- REMOÇÃO:
  - Dados Estáticos – apenas o conteúdo do nó é deletado;
  - Dados Dinâmicos – o nó é eliminado completamente da estrutura.
- INSERÇÃO:
  - Dados Estáticos – não é possível;
  - Dados Dinâmicos – um nó é adicionado à estrutura, aumentando o número de elementos da mesma.

Listas Lineares

- Estrutura caracterizada por uma seqüência ordenada de nós, no sentido de sua posição relativa:

$$E_0,\ E_1,\ E_2,\ \ldots,\ E_n$$

Na memória:



Lista Linear

Listas Lineares

- Regras:

1. Existem (n+1) elementos na seqüência;
2. Eo é o primeiro elemento da seqüência;
3. En é o último elemento da seqüência;
4. Para
   $\forall$ i/ 0 $\leq$ i $\leq$ n
   e j/ 0 $\leq$ j $\leq$ n
   Se i<j $\Rightarrow$ E_i antecede E_j e E_j sucede E_i
5. Se i=j-1 $\Rightarrow$ E_i é o antecessor de E_j e E_j é o sucessor de E_i.

Listas Lineares

- Exemplos de lista linear em C:
  - Acesso Sequencial, dado homogêneo. Ex:

```
// declara lista de 20 nós, contendo words de 4 bits
    int  Lista_words [20][4];
    ...
    Lista_words[i][0] = 1; // deu valor 1011
    Lista_words[i][1] = 1; //para o nó i
    Lista_words[i][2] = 0; // que é uma palavra
    Lista_words[i][3] = 1; // de 4 bits
    i++;
```

Listas Lineares

- Acesso Sequencial, dado heterogêneo. Ex:

```
// declara lista de 20 nós, contendo words de 4 bits
    struct words
    {char B0;
      char B1;
      char B2;
      char B3;
     } Lista_words[20];
    ...
    Lista_words[i].B0 = 1; // deu valor 1011
    Lista_words[i].B1 = 1; //para o nó i
    Lista_words[i].B2 = 0; // que é uma palavra
    Lista_words[i].B3 = 1; // de 4 bits
    i++;
```

Listas Lineares

- Acesso Encadeado, dado homogêneo. Ex:

```
// declara lista de 20 nós, contendo words de 4 bits
    int  Lista_words [20][4], *ptr=
     &Lista_words[0][0];
    ...
    ptr +=i*4; //acessa o i-ésimo nó
    *ptr++ = 1; // deu valor 1011
    *ptr++ = 1; //para o nó i
    *ptr++ = 0; // que é uma palavra
    *ptr = 1; // de 4 bits
    ptr++;
```

Listas Lineares

- Acesso Encadeado, dado heterogêneo. Ex:

```
// declara lista de 20 nós, contendo words de 4 bits
    struct words
    {  char B0;
      char B1;
      char B2;
      char B3;
      struct words * ptr;
     } * prim, * atual;
    ... cont=0;
    atual = prim;
    while(cont < i)
    {   atual = atual->ptr;
        cont++;   }
    atual->B0 = 1; // deu valor 1011
    atual->B1 = 1; //para o nó i
    atual->B2 = 0; // que é uma palavra
    atual->B3 = 1; // de 4 bits
```

Listas Lineares

- Percurso na Lista Linear:
  - O primeiro elemento a ser acessado é o primeiro elemento da lista (E_0);
  - Para acessar-se o elemento E_i, todos os elementos de E_0 até E_{i-1} já foram acessados;
  - O último elemento a ser acessado é o último elemento da lista linear.
- Busca na Lista Linear:
  - A identificação do elemento na lista pode ser feita pela sua posição relativa na lista (índice), ou
  - Por seu valor/conteúdo.

Listas Lineares

- Algoritmo de Busca – Alocação Sequencial:

```
#define N 50
struct x1
{ int chave; //campo que define a busca
  ...} Lista[N], *PtrIni, *PrtAtual,
  *PtrFim=PtrIni;
  ...
struct x1* busca1(int num)
{ PtrAtual = PtrIni;
while((PtrAtual->chave != num)&&(PtrAtual <=
  PtrFim)) PtrAtual++;
if(PtrAtual <= PtrFim) return PtrAtual;
else return NULL;
```

Listas Lineares

- Algoritmo de Busca – Alocação Encadeada:

```
struct x2
{ int chave; //campo que define a busca
  struct x2 * PProx;
} *PtrIni, *PrtAtual;
  ...
struct x2* busca(int num)
{ PtrAtual = PtrIni;
while((PtrAtual->chave != num)&&(PtrAtual->PProx
  != NULL)) PtrAtual=PtrAtual->PProx;
if(PtrAtual->PProx != NULL) return PtrAtual;
else return NULL;
}
```

Listas Lineares

- Algoritmo de Busca – Aloca

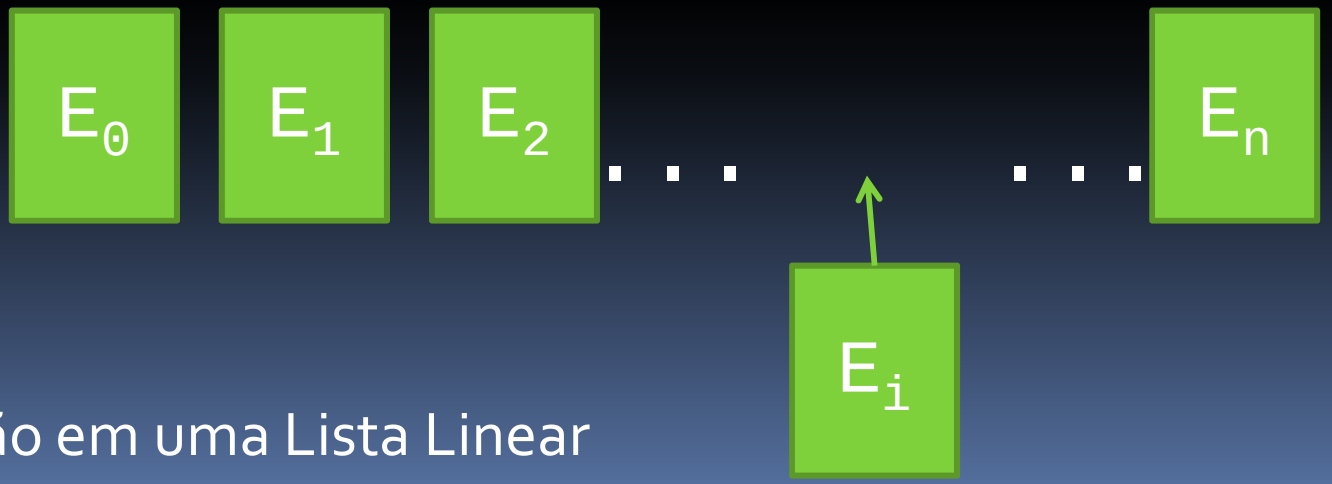
```
struct x2
{ int chave; //campo que d
  struct x2 * PProx;
} *PtrIni, *PrtAtua
  ...
struct x2* busca
{ PtrAtual = PtrIn
while((PtrAtual->cha   !=
  != NULL)) PtrAtual=P rAt
if(PtrAtual->PProx != NULL
else return NULL;
}
```

ox

Expanda o algoritmo da Livraria para fazer busca de um livro pela chave "Titulo"

Listas Lineares

- Inserção na Lista Linear (dinâmica):
  - Se o elemento X for inserido na posição i da lista (com $0 \leq i \leq n$), X para a ser o i-ésimo elemento da lista linear.
  - A lista passa a ter (n+2) elementos.



Inserção em uma Lista Linear

Listas Lineares

- Algoritmo de Inserção– Alocação Sequencial:

```
void insere1(struct x1* novodado, int posicao)
{ if((PtrFim+1)< &Lista[N])
       PtrAtual = PtrFim + 1;
  while(PtrAtual > (PtrIni + posicao))
  {  PtrAtual->chave = (PtrAtual-1)->chave;
     PtrAtual->... = (PtrAtual-1)->...;
     PtrAtual--;
  }
  PtrAtual->chave = novodado->chave;
  PtrAtual->... = novodado->...;
  PtrFim++;
}
```

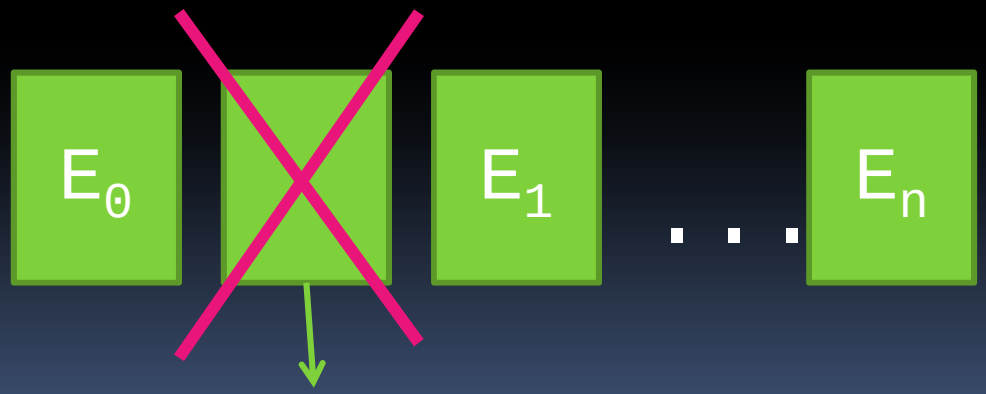
Listas Lineares

- Algoritmo de Inserção – Alocação Encadeada:

```
void insere2 (struct x2* Ptrnovo,int posicao)
{ int i=1;
  PtrAtual = PtrIni;
  while(i++ < (posicao-1))
     PtrAtual = PtrAtual->Prox;
  Ptrnovo->Prox = PtrAtual->Prox;
  PtrAtual->Prox = Ptrnovo;
 }
```

Listas Lineares

- Remoção da Lista Linear (dinâmica):
  - Se o elemento X for removido da posição i da lista, seu sucessor para a ser o sucessor do seu antecessor. Isto é: $E_0, E_1, \ldots, E_{i-1}, E_{i+1}, \ldots, E_n$
  - A lista passa a ter (n+1) elementos.



Remoção em uma Lista Linear

Listas Lineares

- Algoritmo de Remoção– Alocação Sequencial:

```
void remove1(struct x1* PtrRemov)
PtrAtual = PtrRemov;
{while(PtrAtual <= PtrFim)
  {  PtrAtual->chave = (PtrAtual+1)->chave;
     PtrAtual->... = (PtrAtual+1)->...;
     PtrAtual++;
  }
  PtrFim--;
}
```

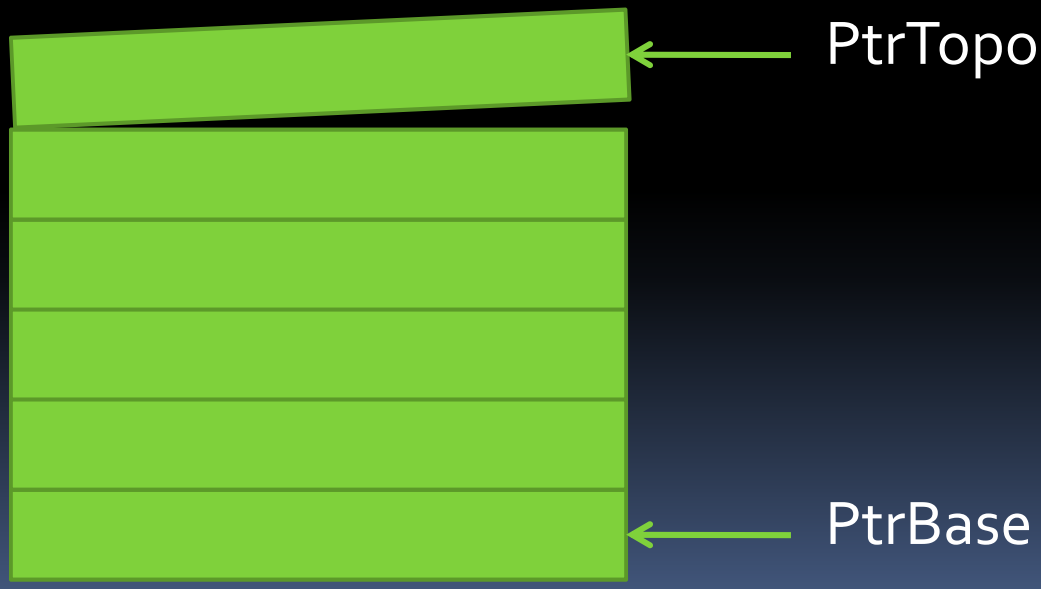
Listas Lineares

- Algoritmo de Remoção – Alocação Encadeada:

```
void remove2 (struct x2* PtrRemov)
{ int i=1;
   PtrAtual = PtrIni;
  while(PtrAtual->Prox != PtrRemov)
     PtrAtual = PtrAtual->Prox;
  PtrAtual->Prox = PtrRemov->Prox;
}
```

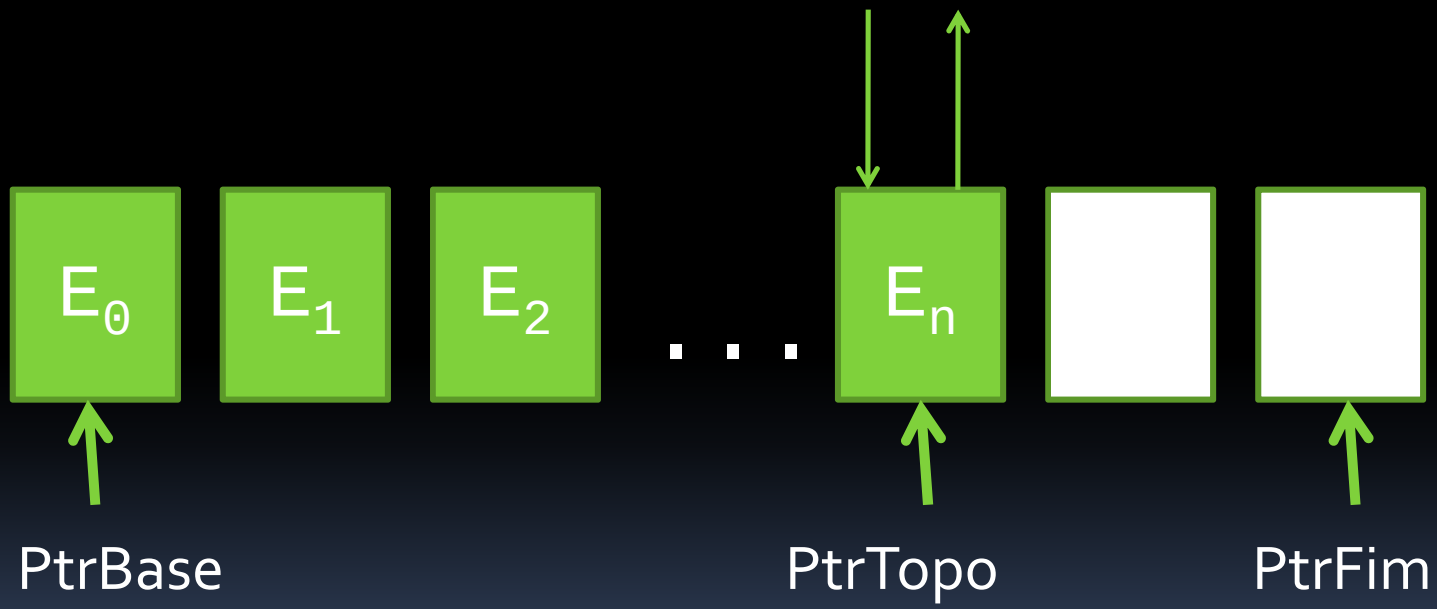
Pilhas

- Uma pilha, ou "stack", é um tipo particular de lista linear no qual as operações de inserção e remoção somente são efetuadas no final, ou "Topo", da lista.



Pilhas

- Pilhas são estruturas do tipo LIFO ("last in, first out")



Lista Linear do tipo Pilha

Pilhas

- Algoritmo de Inserção – Alocação Sequencial:

```
void empilha(struct x1* novodado)
{ if(PtrTopo < PtrFim) //ou < &Lista[N]
  { PtrTopo++;
    PrTopo->chave = novodado->chave;
    PtrTopo->...  = novodado->...;
  }
  else printf("\nOverflow!");
}
```

Pilhas

- Algoritmo de Remoção – Alocação Sequencial:

```
void desempilha()
{ if(PtrTopo >= PtrBase) //ou >= &Lista[0]
  {  PrTopo->chave = 0;  // opcional
     PtrTopo->...  = 0;  // opcional
     PtrTopo--;
  }
   else printf("\nUnderflow!");
}
```

Pilhas

- Algoritmo de Inserção – Alocação Encadeada:

```
void empilha2 (struct x2* Ptrnovo)
{ struct x2 * Ptrtemp;
  PtrTopo->Prox = Ptrnovo;
  PtrTopo = Ptrnovo;
}
```

Pilhas

- Algoritmo de Remoção – Alocação Encadeada:

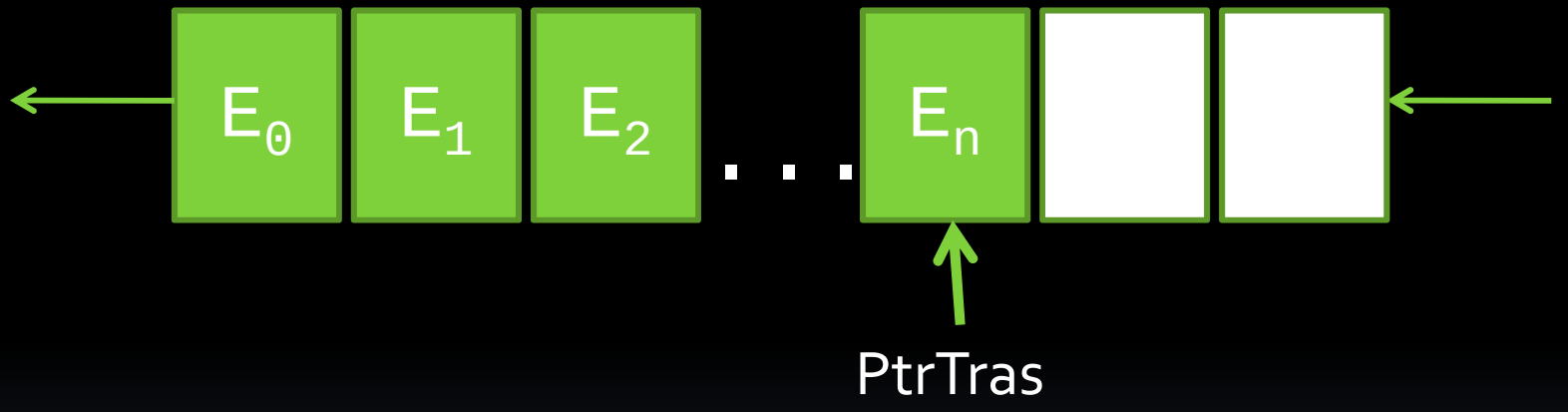
```
void desempilha2 ()
{   PtrAtual = PtrBase;
  while(PtrAtual->Prox != PtrTopo)
     PtrAtual = PtrAtual->Prox;
  PtrTopo = PtrAtual;
  PtrTopo->Prox = NIL;
}
```

Filas

- Uma fila, ou "queue", é um tipo de lista linear no qual a operação de inserção é efetuada no final e a remoção, no início da lista.
- Isto é:
  - A inserção do elemento X na lista torna-o o último da lista linear;
  - A remoção é efetuada sobre o elemento E_0.

Filas

- Filas são estruturas do tipo FIFO ("first in, first out")



Lista Linear do tipo Fila

Filas

- Algoritmo de Inserção – Alocação Sequencial:

```
void enfileira(struct x1* novodado)
{ if(PtrTras < &Lista[N])
  { PtrTras++;
    PrTras->chave = novodado->chave;
    PtrTras->...  = novodado->...;
  }
  else printf("\nOverflow!");
}
```

Filas

- Algoritmo de Remoção – Alocação Sequencial:

```
void desenfileira()
{  if(PtrFrente != NULL)
  {  PtrAtual = PtrFrente;
     while(PtrAtual < PtrTras)  //adianta os nós
     { PtrAtual->chave = (PtrAtual+1)->chave;
       PtrAtual->... = (PtrAtual+1)->...;
       PtrAtual++;
     }
     PtrTras--;
  }
   else printf("\nUnderflow!");
}
```

Filas

- Algoritmo de Inserção – Alocação Encadeada:

```
void enfileira2 (struct x2* Ptrnovo)
{
  PtrTras->Prox = Ptrnovo;
  PtrTras = Ptrnovo;
}
```

Filas

- Algoritmo de Remoção – Alocação Encadeada:

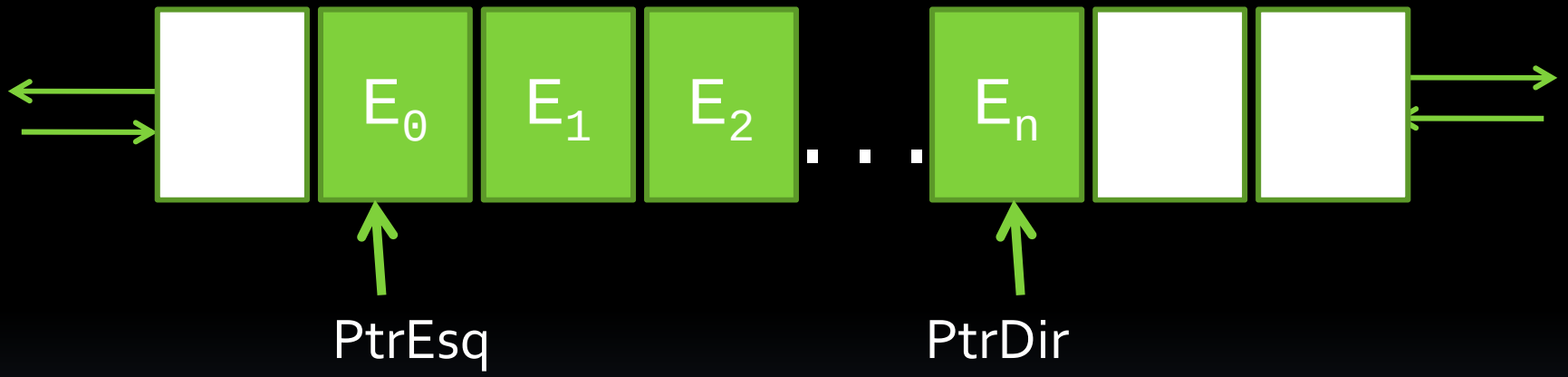
```
void desenfileira2 ()
{ if(PtrFrente!=PtrTras)
   PtrFrente= PtrFrente->Prox;
  else printf("\nUnderflow!");
}
```

Deque

- Um deque, ou "double-ended queue", é um tipo de lista linear no qual as operações de e remoção podem ser efetuadas tanto no início quanto no final da lista linear.
- Isto é:
  - A inserção do elemento X na lista torna-o o primeiro ou último da lista linear – funções *insereesq*() e *inseredir*();
  - A remoção é sempre efetuada sobre o elemento E_0 ou sobre o E_m - funções *removeesq*() e *removedir*();

Deque

- Deques são o caso mais geral das filas e pilhas.



Lista Linear do tipo Deque

Deque

- Algoritmo de Inserção à Esquerda – Alocação Sequencial:

```
void insereesq(struct x1* novodado)
{ if( PtrEsq > &Lista[0])
  {  PtrEsq--;
     PtrEsq->chave = novodado->chave;
     PtrEsq->... = novodado->...;
  } else printf("\nOverflow!");
}
```

Deque

- Algoritmo de Inserção à Direita – Alocação Sequencial :

```
void inseredir (struct x1* novodado)
{  if(PtrDir < &Lista[N])
  {  PtrDir++;
     PtrDir->chave = novodado->chave;
     PtrDir->... = novodado->...;
  } else printf("\nOverflow!");
}
```

Deque

- Algoritmo de Remoção à esquerda – Alocação Sequencial:

```
void removeesq()
{if(PtrDir >= PtrEsq)
 { while(PtrAtual <= PtrDir)
  {  PtrAtual->chave = (PtrAtual+1)->chave;
     PtrAtual->... = (PtrAtual+1)->...;
     PtrAtual++;
  }
  PtrDir--;
 else printf("\nUnderflow!");
}
```

Deque

- Algoritmo de Remoção à direita – Alocação Sequencial:

```
void removedir()
{  if(PtrDir >= PtrEsq)
  {  PtrAtual = PtrDir;
     while(PtrAtual > PtrEsq)  //atrasa os nós
     { PtrAtual->chave = (PtrAtual-1)->chave;
       PtrAtual->... = (PtrAtual-1)->...;
       PtrAtual--;
     }
     PtrEsq++;
  }
   else printf("\nUnderflow!");
}
```

Deque

- Algoritmo de Inserção à Esquerda – Alocação Encadeada:

```
void insereesq2(struct x2* Ptrnovo)
{
     Ptrnovo->Prox = PtrEsq;
     PtrEsq = Ptrnovo;
}
```

Deque

- Algoritmo de Inserção à Direita – Alocação Encadeada:

```
void inseredir (struct x2* Ptrnovo)
{   PtrDir->Prox = Ptrnovo;}
```

Deque

- Algoritmo de Remoção à esquerda – Alocação Encadeada:

```
void removeesq()
{if(PtrEsq != NULL)
  PtrEsq = PtrEsq->Prox;
 else printf("\nUnderflow!");
}
```

Deque

- Algoritmo de Remoção à direita – Alocação Encadeada:

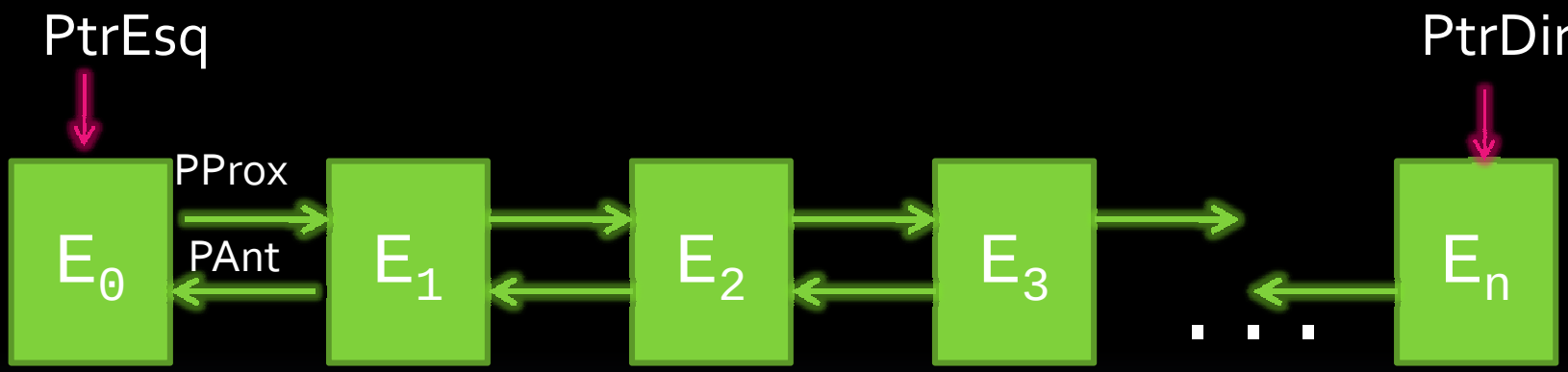
```
void removedir()
{  if(PtrEsq !=NULL)
  {  PtrAtual = PtrEsq;
     while(PtrAtual->Prox != PtrDir)
          PtrAtual = PtrAtual->Prox;
     PtrDir = PtrAtual;
     PtrAtual->Prox = NULL;
  }
   else printf("\nUnderflow!");
}
```

Lista duplamente encadeada

- Percebe-se que, para ter acesso ao nó anterior ao nó atual (para remoção ou inserção), temos que usar um ponteiro rastreador (PtrAtual).
- Como ter acesso imediato a cada nó anterior?
- Para solucionar o problema: mais um link em cada nó, de forma que, além de apontar para o sucessor, o nó aponte também para seu antecessor

- LISTA DUPLAMENTE ENCADEADA

Lista duplamente encadeada



Lista duplamente encadeada

TRABALHO 1:

- Implemente o programa que encapsula todas as funcionalidades de uma lista linear duplamente encadeada, quais sejam:

  1) Inicializar a lista;

  2) Inserir nó com a chave NOME ordenada alfabeticamente;

  3) Buscar nó pela chave NOME(devolve endereço do nó);

  4) Remover nó do endereço x;

  5) Alterar nó do endereço x;

  6) Listar todos os nós.

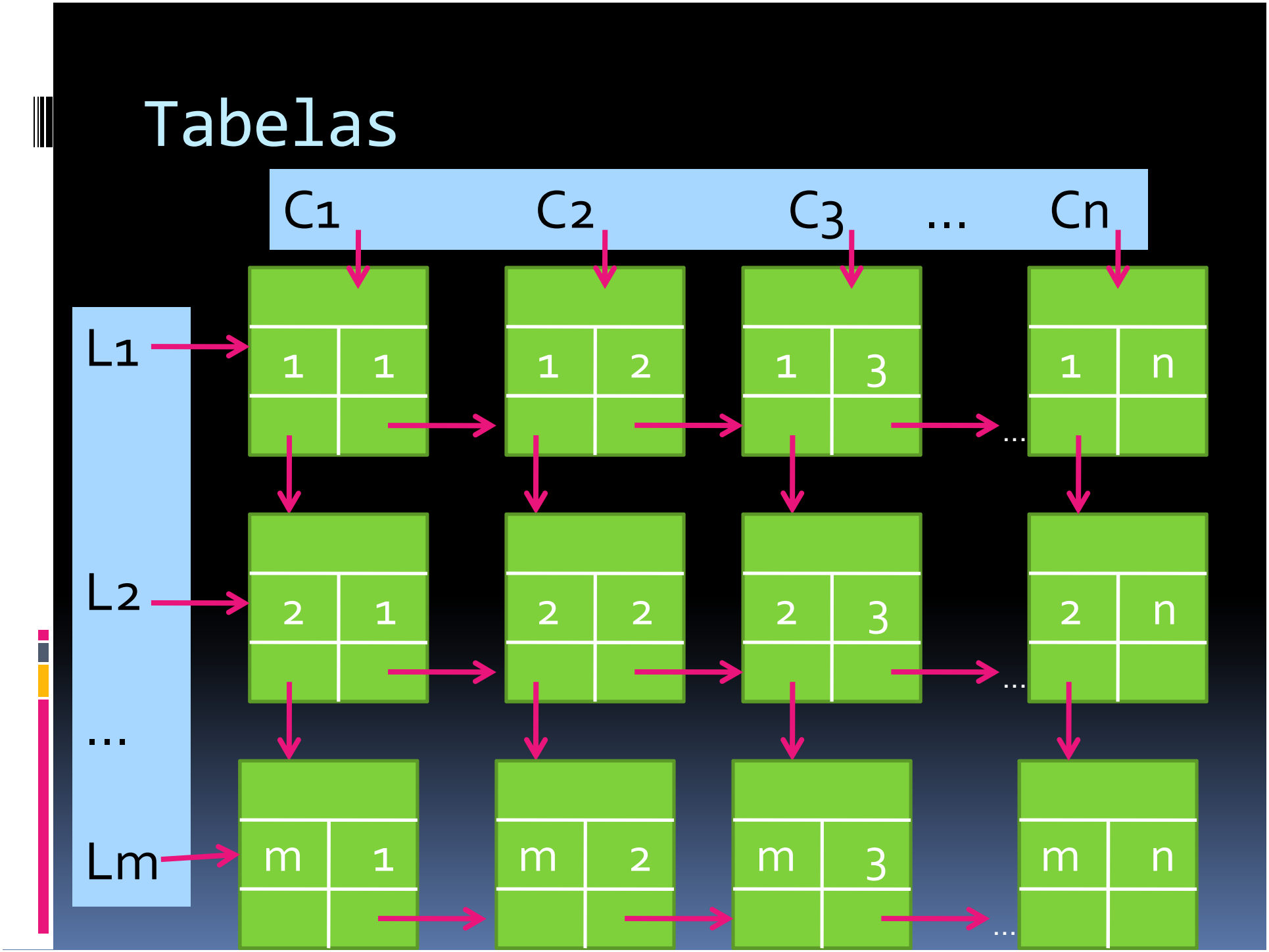
  - Data de Entrega: 21/05/10

{String NOME}
String Telefone
int idade

Tabelas

- Ao contrário das listas lineares, tabelas são estruturas 2D.
- Ou seja, para acessar-se um determinado nó, são necessários duas posições, dois ponteiros externos – um para a "Linha" e outro para a "Coluna" onde está localizado o nó.
- Cada nó será então:

Tabelas



Árvores

- Estruturas de dados não-lineares e que incluem informação de hierarquia e ordenação.
- Pode ser definida como conjunto finito, de um ou mais nós, tais que:

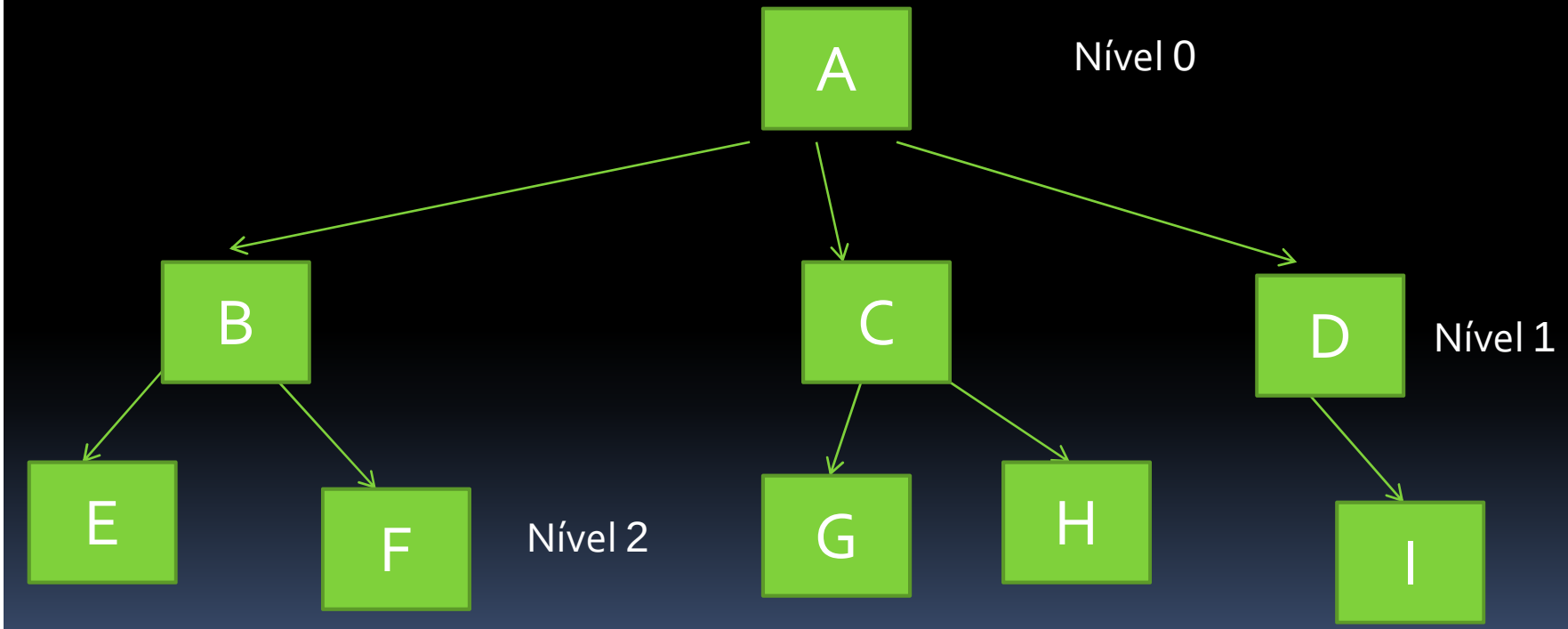
1. Existe um nó definido como raiz da árvore;
2. Os demais nós formam M >= 0 conjuntos disjuntos S1, S2, ... , Sm, onde cada um desses subconjuntos é uma árvore, ou "subárvore".

Árvores

- Ex: árvore genealógica

Árvores

- Outro conceito importante em estruturas de árvore é o de nível – distância do nó à raiz.



Árvores

- Quanto à ordenação dos filhos de um nó, uma árvore é:
  - Não-ordenada – onde a ordem dos filhos é irrelevante para a aplicação. Neste caso, apenas a hierarquia que a estrutura proporciona é relevante;
  - Ordenada – a ordem dos filhos é relevante.

Árvores