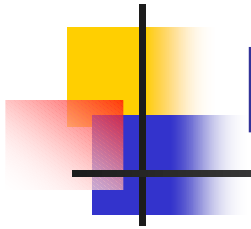
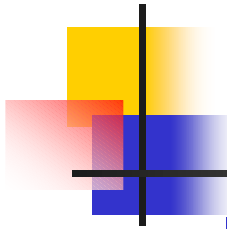


Listas

- Uma lista é uma seqüência de 0 ou mais elementos de um determinado tipo.
- Representação:

  $a_1, a_2, \ldots, a_n$

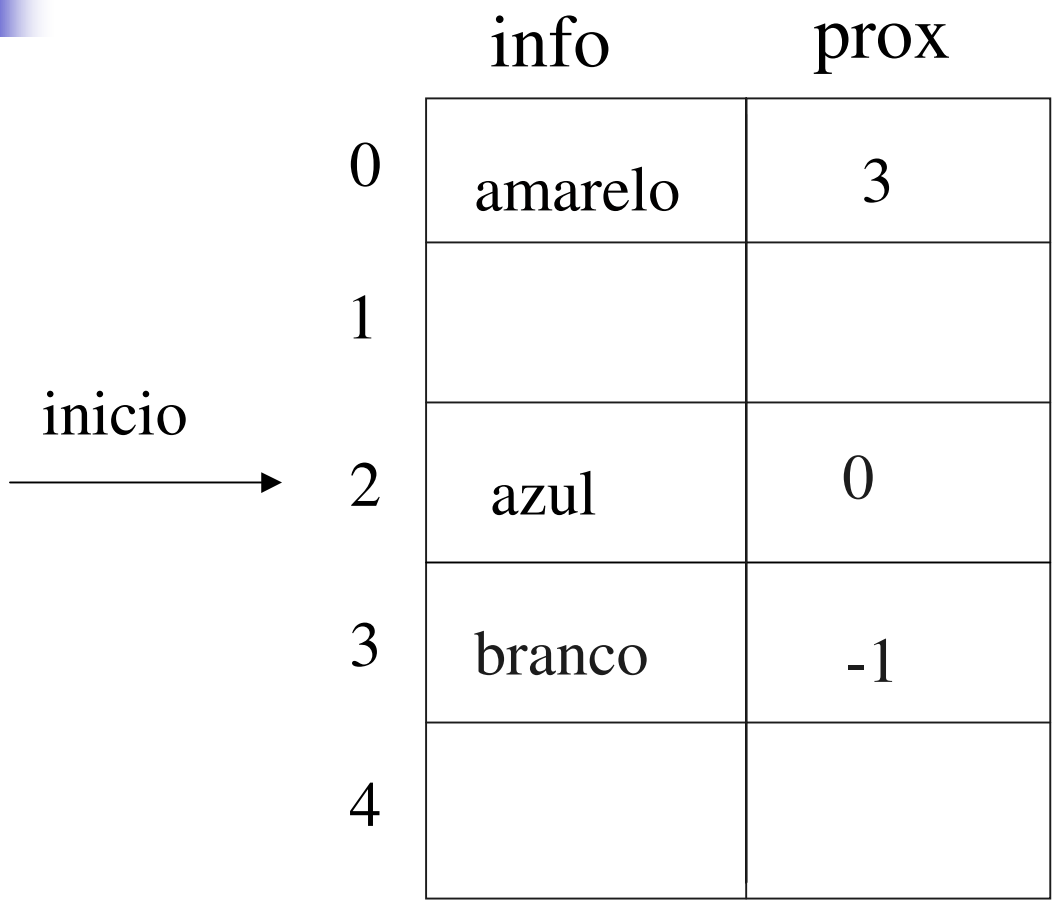
  $n \geq 0$ e a_i têm mesmo tipo
  - n é o tamanho da lista
  - elemento a_i está na posição i
  - elementos são ordenados (a_i precede a_{i+1})

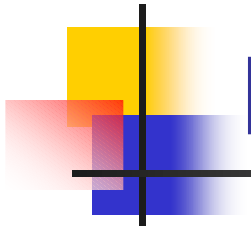


Listas Ligadas - Operações

- insere(x,p.L): insere x na posição p da lista L movendo elementos a partir de p para uma posição adiante na lista.
- busca(x,L): retorna posição de x na lista L.
- recupera(p,L): retorna elemento na posição p da lista L.
- remove(p,L): remove elemento na posição p da lista L.
- imprime(L): imprime elementos de L na ordem de ocorrência.

Listas Simples
Representação com vetor



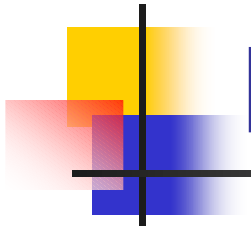


Listas Simples - vetor (1)

```
#define MAX 100
struct no {
   char info[30];
   int prox;
};
struct no vno[MAX];
int livre,inicio;
```

Listas Simples - vetor (2)

```
/* constroi lista de espaço livre */
void inicia ()
{   int i;
    for (i=0;i<MAX-1;i++)
        vno[i].prox=i+1;
    vno[MAX-1].prox=-1;
    livre = 0;
}
```

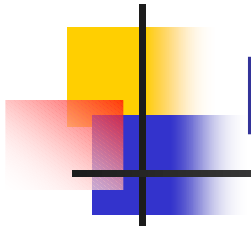


Listas Simples - vetor (3)

```
/* insere nó i na lista de espaço livre */
void libera_no(int i)
{
   vno[i].prox=livre;
     livre=i;
}
```

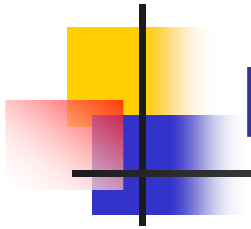
Listas Simples - vetor (4)

```
int obtem_no()
{
   int aux;
   if (livre==-1) {
         printf("estouro\n"); exit(1);
    }
    aux=livre;
    livre=vno[livre].prox;
    return(aux);
}
```



Listas Simples - vetor (5)

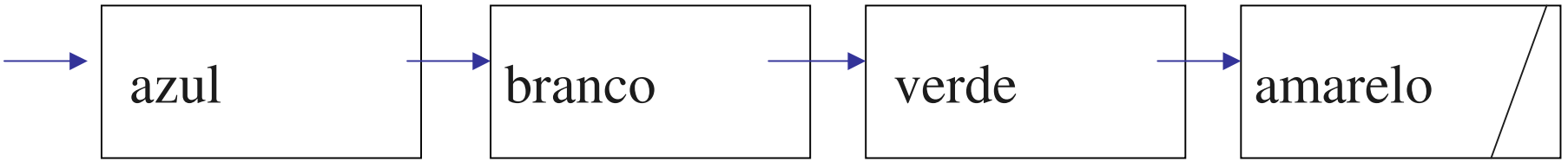
```
int insere(int p, char x[])
/* insere após p */
{    int q;
     if (p==-1) {
          printf("falha insercao\n")';return(0);
     }
     q=obtem_no();
     strcpy(vno[q].info,x);
     vno[q].prox=vno[p].prox;
     vno[p].prox=q;
     return(1);
}
```

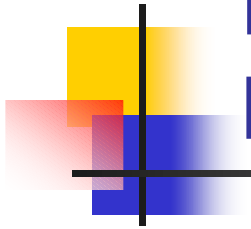


Listas Simples - vetor (6)

```
int remove(int p, char x[])
/* remove nó após p */
{   int q;
    if ( (p==-1)||(vno[p].prox==-1)) {
        printf("falha emocao\n");return(0);
    }
    q=vno[p].prox; vno[p].prox=vno[q].prox;
    strcpy(vno[p].info,x);
    libera_no(q);
    return(1);
}
```

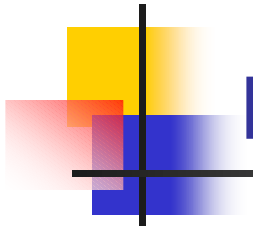
Listas Simples
Representação com apontadores





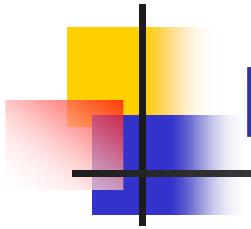
Listas Simples
Declaração

```
struct no {
   char info[30];
   struct no *prox;
};
```



Listas Simples - apontadores (1)

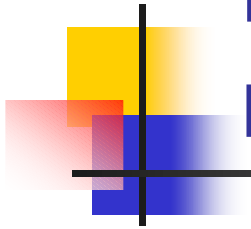
```
/* insere no início da lista */
struct no *insere(char *x, struct no *ini)
{
    struct no * aux;
    aux = (struct no *) malloc(sizeof(struct no));
    strcpy(aux->info,x);
    aux->prox=ini;
    return(aux);
}
```



Listas Simples - apontadores (2)

```
/* busca x, retorna apontador */
struct no * busca (char *x, struct no *ini)
{
    while (ini)
        if(strcmp(ini->info,x))
            ini=ini->prox;
        else return(ini);
    return(NULL);
}
```

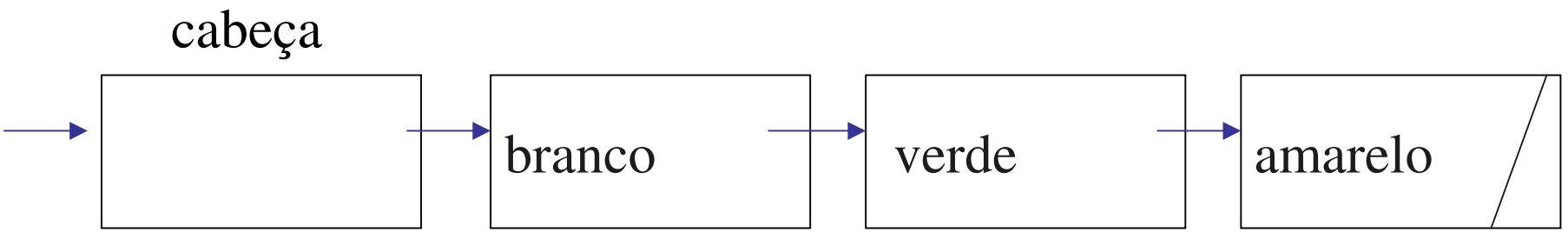
Listas Simples - Comparação das representações



- alocação estática
  - tamanho fixo em tempo de compilação
  - espaço máximo é alocado durante toda execução do programa
- alocação dinâmica
  - custo de alocação para cada nó da lista (na inserção)
  - custo de desalocação de um nó (na remoção)
  - espaço é alocado a medida que o programa precisa

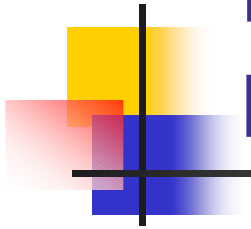
Listas Simples com nó cabeça

- Cabeça: não contém informação. Só tem apontador para início da lista.
- vantagem: evita teste para verificar se é início da lista



Listas Simples com nó cabeça

```
struct no *inicia()
{   struct no *inicio= (struct no *)malloc(sizeof(struct
    no));
     inicio->prox=NULL;
     return(inicio);
}
void insere(struct no *inicio, char *x)
{   struct no *temp=(struct no*)malloc(sizeof(struct
    no));
     strcpy(temp->info,x);
     temp->prox=inicio->prox;
     inicio->prox=temp;
}
```



Listas Simples
Exercícios

- Função para imprimir campo info dos elementos da lista
- Função para contar número de nós na lista
- Função para contar ocorrências do elemento x na lista
- Função para verificar se 2 listas são iguais (retorna 1 ou 0)
- Função para unir 2 listas (retorna apontador para lista unida)
- Função que insere elemento em lista ordenada deixando a lista ordenada
- Função para remover um elemento de uma lista ordenada

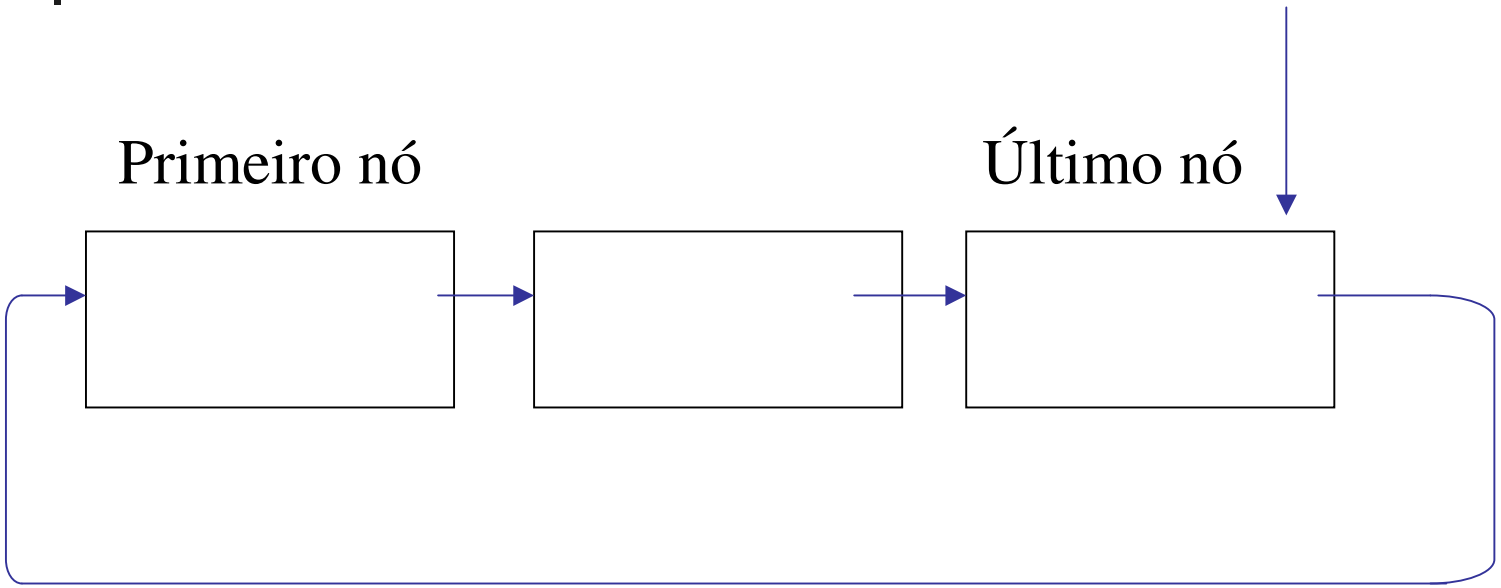
Listas Simples - inserir em lista ordenada

```
void insereordenado(struct no *inicio, char *x)
{   struct no *novo,*atual,*pred;
    atual=inicio->prox; pred=inicio;
    while (atual)
        if (strcmp(x,atual->info)>0)
        {   pred=atual; atual=atual->prox; }
        else break;
    novo= (struct no*)malloc(sizeof(struct no));
    strcpy(novo->info,x); novo->prox=pred->prox;
    pred->prox=novo;
}
```

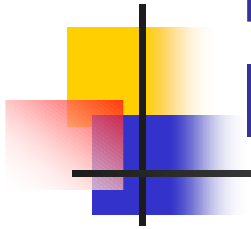
Listas Simples - remover de lista ordenada

```
void removeordenado(struct no *inicio, char *x)
{   struct no *atual,*pred;
    int i;
    atual=inicio->prox; pred=inicio;
    while (atual)
        if (i=strcmp(x,atual->info)>0)
            {   pred=atual; atual=atual->prox;}
        else if (i) return();
            else { pred->prox=atual->prox;
                free(atual);return();}
}
```

Listas Circulares

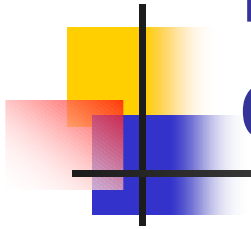


Listas Circulares - inserir no fim da lista

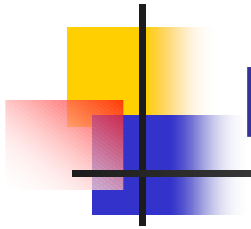


```
struct no *inserecircular(struct no *lista, int x)
{   struct no *novo;
    novo=(struct no *)malloc(sizeof(struct no));
    novo->info=x;
    if(lista){
        novo->prox=lista->prox;
        lista->prox=novo;
    }
    else novo->prox=novo;
    return(novo);
}
```

Listas Circulares - remover primeiro da lista



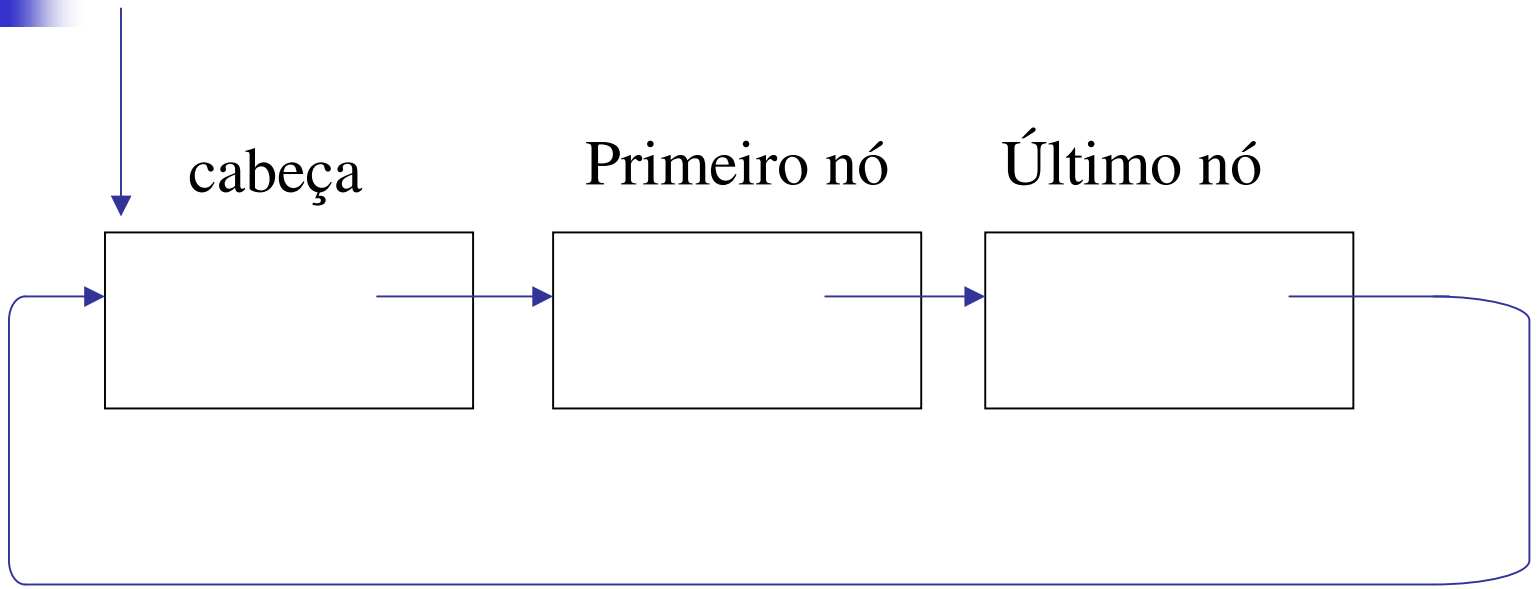
```
struct no *removecircular(struct no *lista)
{   struct no *aux;
    if (!lista) return(NULL);
    aux=lista->prox;
    if(aux==lista) {
       free(aux);return(NULL);
    }
    else {
       lista->prox=aux->prox;free(aux); return(lista);
    }
}
```



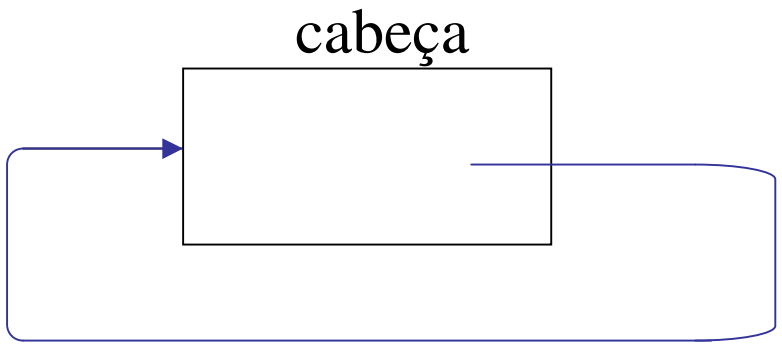
Listas Circulares - buscar na lista

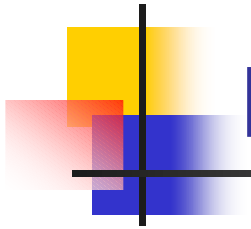
```
struct no *buscacircular(struct no *lista, int x)
{   struct no *aux;
    if (!lista) return(NULL);
    aux=lista->prox;
    while(aux!=lista)
        if(aux->info==x) return(aux);
        else aux=aux->prox;
    if (aux->info==x) return(aux);
    return(NULL);
}
```

Listas Circulares com nó cabeça



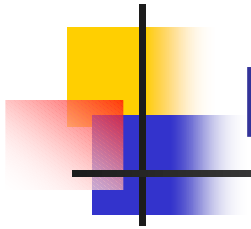
Lista vazia





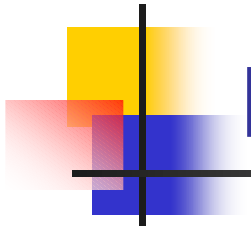
Listas Circulares - inserir no início

```
void inserecircularcab(struct no *cabeca, int x)
{   struct no *novo=(struct no *)
        malloc(sizeof(struct no));
    novo->info=x;
    novo->prox=cabeca->prox;
    cabeca->prox=novo;
}
```



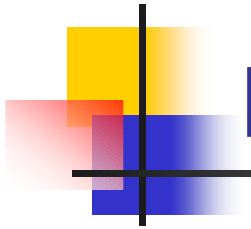
Listas Circulares - remover do início

```
void removecircularcab(struct no *cabeca)
{   struct no *aux;
    if(cabeca!=cabeca->prox)
    {
        aux=cabeca->prox;
        cabeca->prox=aux->prox;
        free(aux);
    }
}
```



Listas Circulares - buscar na lista

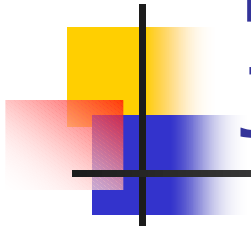
```
struct no *buscacircularcab(struct no *cabeca, int x)
{
    struct no *aux;
    cabeca->info=x; /* sentinela no nó cabeça */
    aux=cabeca->prox;
    while (aux->info!=x)
        aux=aux->prox;
    return((aux==cabeca)?NULL:aux);
}
```



Listas Circulares - Aplicações

- escalonamento de processos
- problema do Josephus: um grupo de soldados cercado pelo exército inimigo que precisa pedir ajuda. Um soldado deve ser escolhido segundo o método
  - um número n é sorteado
  - um nome é sorteado
  - soldados formam um círculo
  - contagem começa pelo soldado sorteado em sentido horário até atingir n. Esse soldade sai do círculo e a contagem reinicia no soldado seguinte.
  - o último soldado que restar deve procurar ajuda.

Listas Circulares - Problema do Josephus



- ler n, nome
- enquanto houver nomes
  - inserir na lista circular
  - ler próximo nome
- enquanto existir mais de um nó na lista
  - andar n-1 nós na lista
  - imprimir n^{ésimo} nó
  - retirar n^{ésimo} nó
- imprimir o nó restante