

Princípios de Desenvolvimento de Algoritmos MAC122

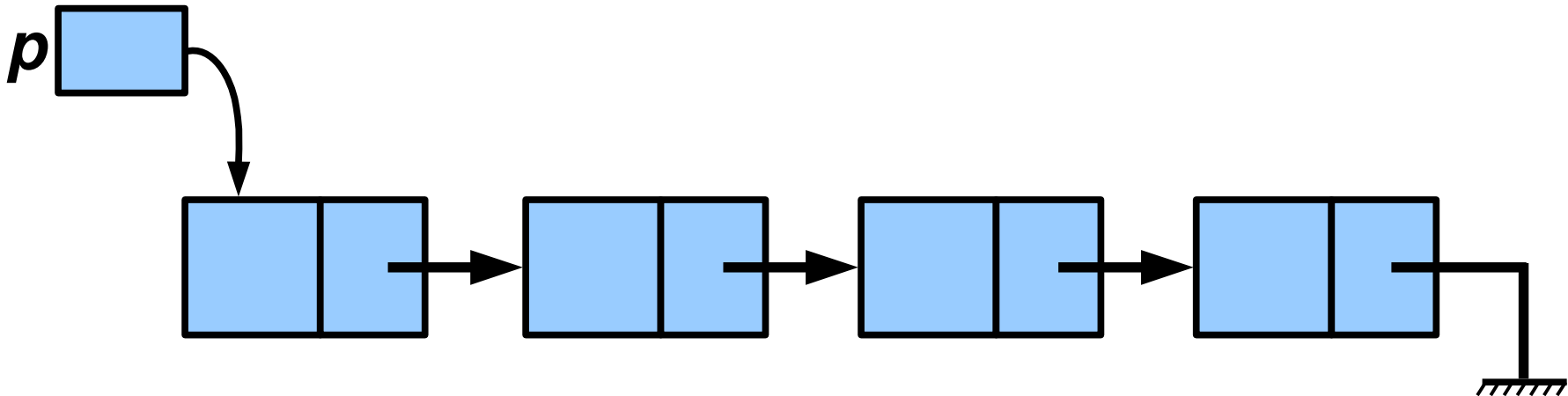
**Prof. Dr. Paulo Miranda
IME-USP**

Listas ligadas

Listas ligadas simples

- Definição típica da estrutura utilizada.

```
struct Reg{  
    int      dado;  
    struct Reg *prox; /* Apontador para  
                       o próximo registro  
                       da lista. */  
};
```



Listas ligadas simples

- Procedimento para inserção de dados.

```
struct Reg *AlocaNoLista() {
    struct Reg *q;
    q = (struct Reg*)malloc(sizeof(struct Reg));
    if (q == NULL)
        exit(1);
    return q;
}

/*O parâmetro p deve apontar para o nó que
precede o nó a ser inserido. */

void InsereAposLista(struct Reg *p, int x) {
    struct Reg *q;

    q = AlocaNoLista();
    q->dado = x;
    q->prox = p->prox;
    p->prox = q;
}
```

Listas ligadas simples

- Procedimento para remoção de dados.

```
/* O parâmetro p deve apontar para o nó que
   precede o nó a ser removido. */

void RemoveAposLista(struct Reg *p) {
    struct Reg *q;

    q = p->prox;
    if (q != NULL) {
        p->prox = q->prox;
        free(q);
    }
}
```

Listas ligadas simples

- Procedimento para remoção de dados.

```
/* O parâmetro p deve apontar para o nó que
   precede o nó a ser removido. */

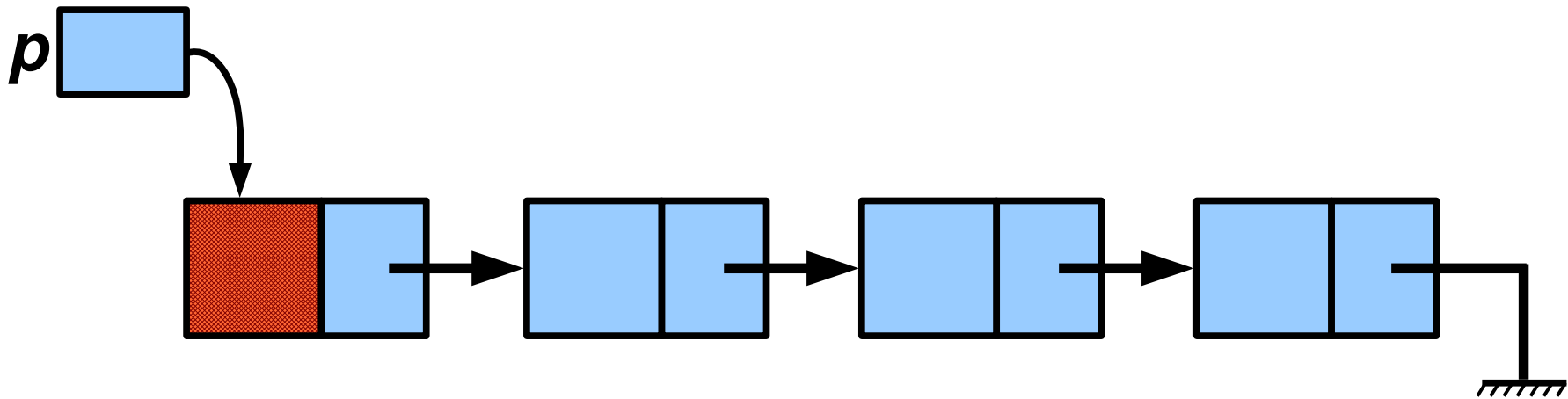
void RemoveAposLista(struct Reg *p) {
    struct Reg *q;

    q = p->prox;
    if (q != NULL) {
        p->prox = q->prox;
        free(q);
    }
}
```

- **Problema:**
 - Os procedimentos não funcionam para o caso em que deseja-se fazer a operação no início da lista.

Listas ligadas simples

- Solução #1:
 - Adotar um primeiro nó virtual, denominado *nó-cabeça*.



Listas ligadas simples

- Solução #1:
 - Adotar um primeiro nó virtual, denominado *nó-cabeça*.

```
struct Reg *CriaNoCabeça() {  
    struct Reg *p;  
    p = AlocaNoLista();  
    p->dado = -1;          /* valor diferenciado. */  
    p->prox = NULL;  
    return p;  
}
```

Listas ligadas simples

- Solução #2: (sem uso de nó-cabeça)
 - Reformular código usando ponteiro duplo (apontador para apontador).
 - A inserção no início se dá passando em **p** o endereço do apontador para o início da lista.
 - Já a inserção após um nó arbitrário apontado por **r**, é possível usando **p = &(r->prox);**

```
void InsereLista(struct Reg **p, int x) {
    struct Reg *q;
    struct Reg *t;

    t = *p;
    q = AlocaNoLista();
    q->dado = x;
    q->prox = t;
    *p = q;
}
```


Listas ligadas simples

- Solução #2: (sem uso de nó-cabeça)
 - Reformular código usando ponteiro duplo (apontador para apontador).
 - A inserção no início se dá passando em **p** o endereço do apontador para o início da lista.
 - Já a inserção após um nó arbitrário apontado por **r**, é possível usando **p = &(r->prox);**

```
void RemoveLista(struct Reg **p) {
    struct Reg *t;

    t = *p;
    if (t != NULL) {
        *p = t->prox;
        free(t);
    }
}
```

Listas ligadas simples

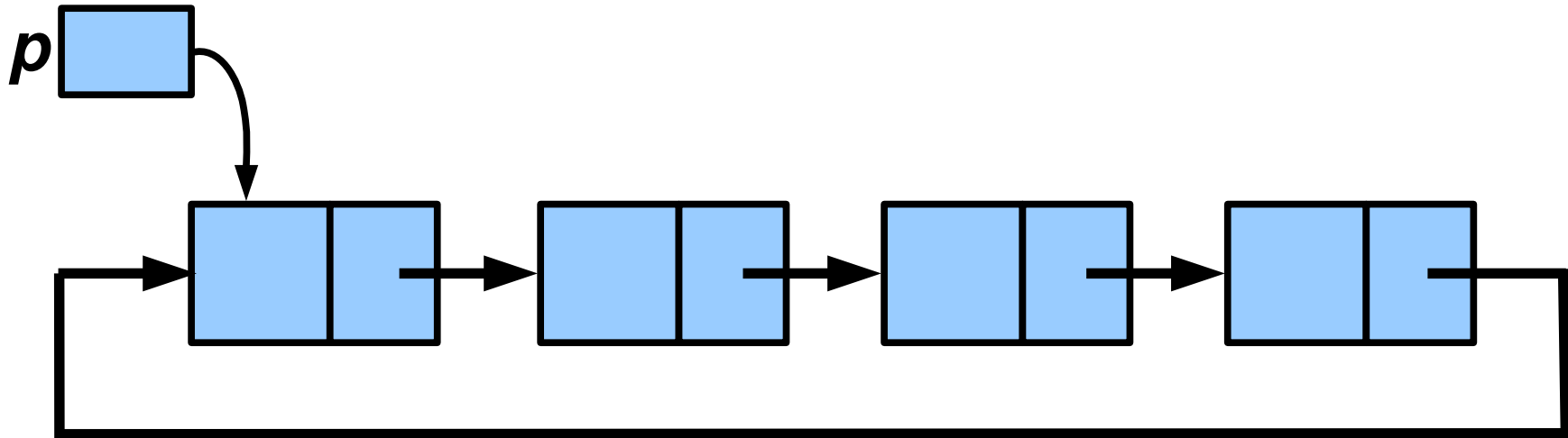
- Exemplo de função que percorre a lista.

```
void ImprimeLista(struct Reg *p) {
    struct Reg *q;

    q = p; /* ou q = p->prox; (pular nó-cabeça).*/
    if (q == NULL) {
        printf("Lista vazia\n");
        return;
    }

    while (q != NULL) {
        printf(" %2d", q->dado);
        q = q->prox;
    }
    printf("\n");
}
```

Listas circulares



- No caso de listas circulares (**não vazias**), para todo nó sempre existe um nó anterior.
- Logo podemos usar sempre as funções 'InsereApos' e 'RemoveApos', mesmo sem necessidade de nó-cabeça.

Listas circulares

- Procedimento para inserção de dados.

```
void InsereAposCircular(struct Reg **p, int x){
    struct Reg *q;
    struct Reg *t;

    q = AlocaNoLista();
    q->dado = x;
    t = *p;
    if (t == NULL) { /* para tratar lista vazia. */
        q->prox = q;
        *p = q;
    }
    else {
        q->prox = t->prox;
        t->prox = q;
    }
}
```

Listas circulares

- Procedimento para remoção de dados.

```
/* *p deve apontar para o nó que
   precede o nó a ser removido. */

void RemoveAposCircular(struct Reg **p) {
    struct Reg *q;
    struct Reg *t;
    t = *p;
    if (t == NULL) return;
    q = t->prox;
    if (q == t)
        *p = NULL;
    else
        t->prox = q->prox;
    free(q);
}
```

Listas circulares

- Exemplo de função que percorre a lista.

```
void ImprimeCircular(struct Reg *p) {
    struct Reg *q;

    if (p == NULL) {
        printf("Lista vazia\n");
        return;
    }
    q = p;
    do{
        printf(" %2d", q->dado);
        q = q->prox;
    } while (q != p);
    printf("\n");
}
```

Listas circulares

- Função de busca em uma lista circular.

```
struct Reg * BuscaCircular(struct Reg *p, int x) {
    struct Reg *q;

    if (p == NULL)
        return NULL;

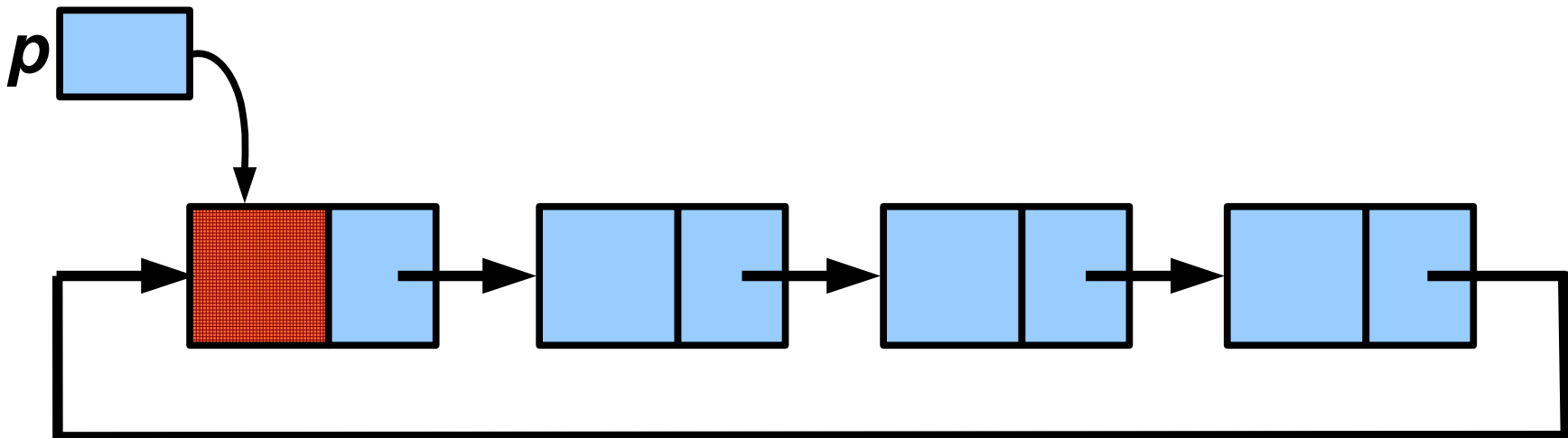
    q = p;
    do{
        if (q->dado == x)
            return q;

        q = q->prox;
    } while (q != p);

    return NULL;
}
```

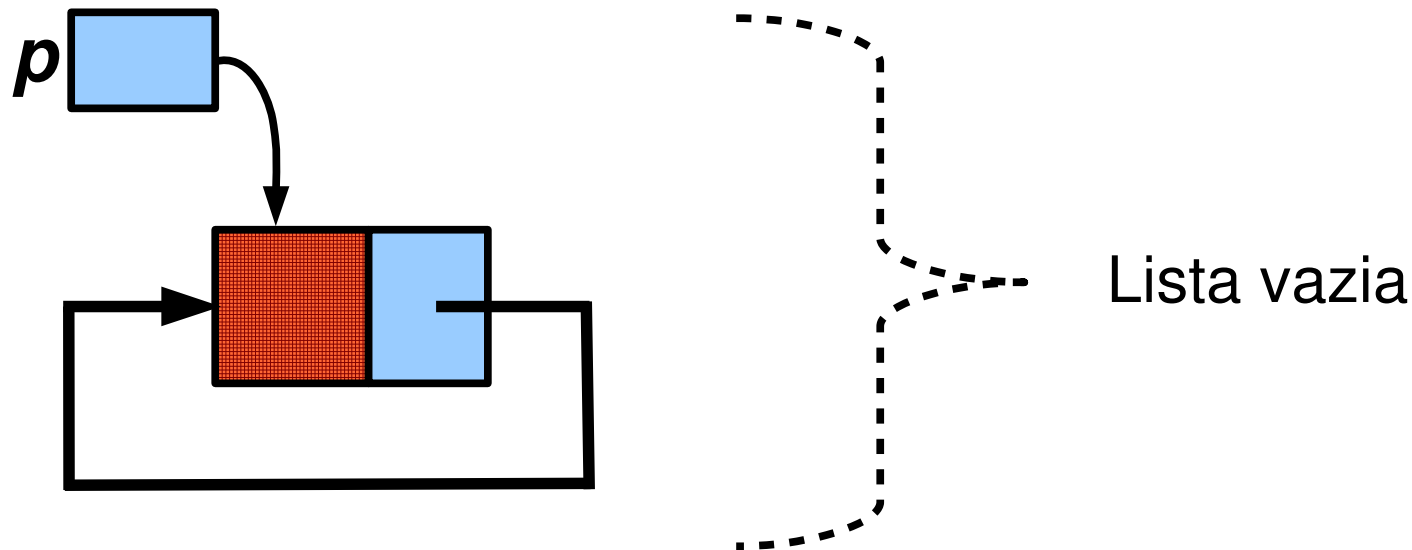
Listas circulares

- A representação anterior adotava valor **NULL** para listas vazias.
- Nos códigos de inserção e remoção, um ponteiro duplo foi usado exclusivamente para tratar listas vazias.
- Este fato está refletido numa certa complicação aparente nos procedimentos anteriores.
- Uma maneira de simplificar os códigos é introduzir, também neste caso, um nó-cabeça:



Listas circulares

- A representação anterior adotava valor **NULL** para listas vazias.
- Nos códigos de inserção e remoção, um ponteiro duplo foi usado exclusivamente para tratar listas vazias.
- Este fato está refletido numa certa complicação aparente nos procedimentos anteriores.
- Uma maneira de simplificar os códigos é introduzir, também neste caso, um nó-cabeça:



Listas circulares

- A representação anterior adotava valor **NULL** para listas vazias.
- Nos códigos de inserção e remoção, um ponteiro duplo foi usado exclusivamente para tratar listas vazias.
- Este fato está refletido numa certa complicação aparente nos procedimentos anteriores.
- Uma maneira de simplificar os códigos é introduzir, também neste caso, um nó-cabeça:

```
struct Reg * CriaNoCabecaCircular() {  
    struct Reg *p;  
  
    p = AlocaNoLista();  
    p->dado = -1;  
    p->prox = p;  
    return p;  
}
```

Listas circulares

- Funções de inserção e remoção com nó-cabeça.

```
void InsereAposCircular(struct Reg *p, int x) {
    struct Reg *q;
    q = AlocaNoLista();
    q->dado = x;
    q->prox = p->prox;
    p->prox = q;
}

/* Tomar cuidado para não remover o nó-cabeça! */
void RemoveAposCircular(struct Reg *p) {
    struct Reg *q;

    q = p->prox;
    if (q != p) {
        p->prox = q->prox;
        free(q);
    }
}
```

Listas circulares

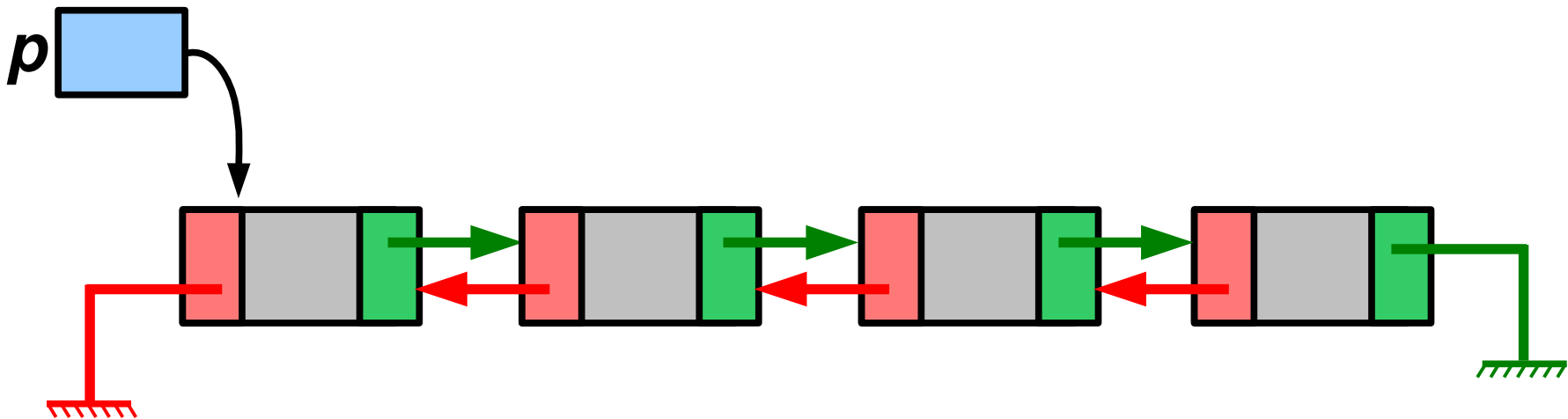
- Função de busca com sentinela em uma lista circular.

```
/* O parâmetro p deve apontar para o nó-cabeça.  
   A função devolve o apontador para o primeiro  
   nó que contém o dado x. */  
  
struct Reg * BuscaCircular(struct Reg *p, int x) {  
    struct Reg *q;  
  
    p->dado = x;  
    q = p;  
  
    do {  
        q = q->prox;  
    } while (q->dado != x);  
  
    p->dado = -1;  
    if (q == p)  
        return NULL;  
    else  
        return q;  
}
```

Listas duplamente ligadas

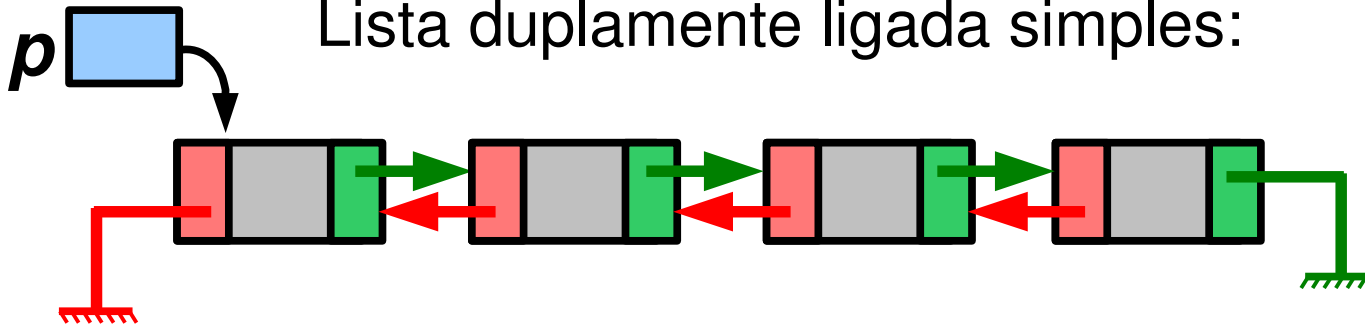
- Definição típica da estrutura utilizada.

```
typedef int TipoDado;  
  
typedef struct _RegDupla{  
    struct _RegDupla *esq;  
    TipoDado          dado;  
    struct _RegDupla *dir;  
} RegDupla;  
  
typedef RegDupla* ListaDupla;  
  
/* 'ListaDupla' é um tipo abstrato que  
   esta sendo implementado. */
```

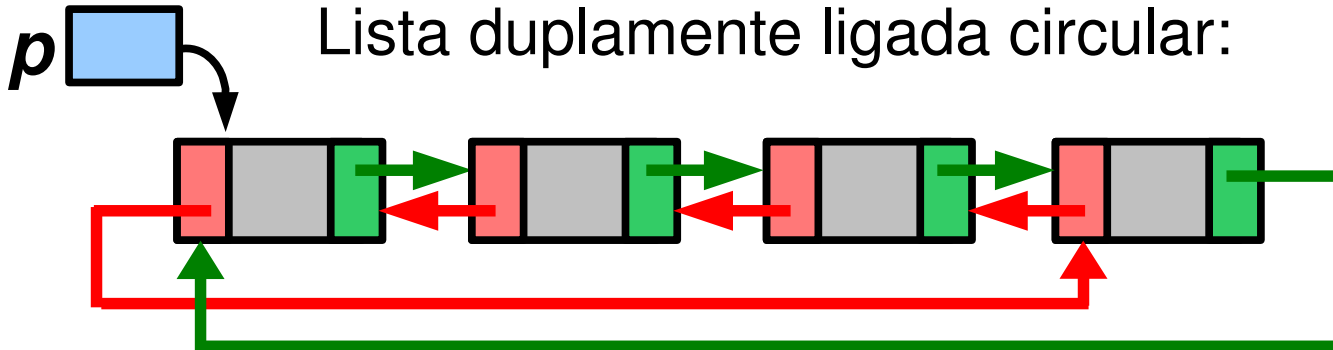


Listas duplamente ligadas

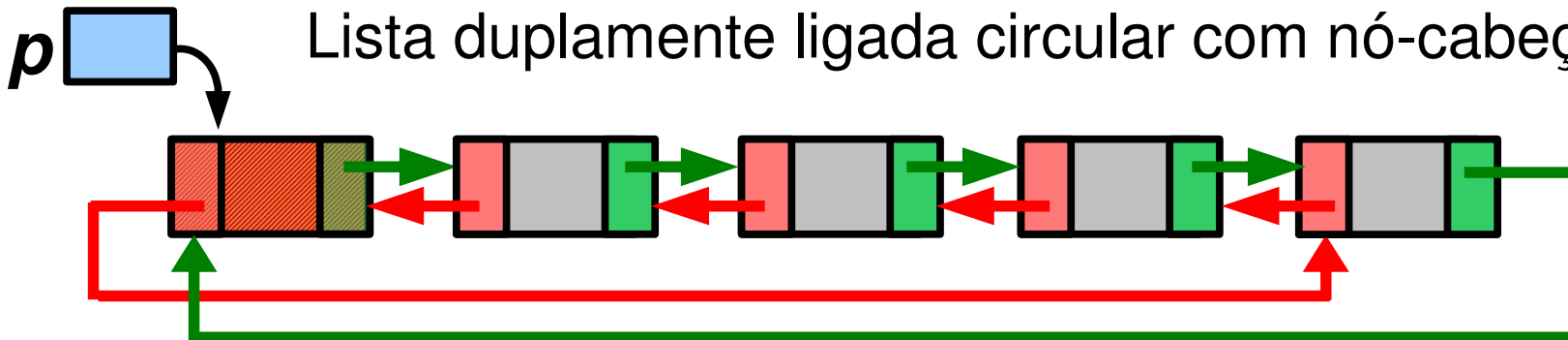
Lista duplamente ligada simples:



Lista duplamente ligada circular:



Lista duplamente ligada circular com nó-cabeça:



Por simplicidade, vamos mostrar apenas o último caso.

Listas duplamente ligadas

- Criação do nó-cabeça.

```
RegDupla* AlocaRegDupla() {  
    RegDupla* q;  
    q = (RegDupla*)calloc(1, sizeof(RegDupla));  
    if(q==NULL) exit(-1);  
    return q;  
}
```

```
ListaDupla CriaNoCabecaDupla() {  
    RegDupla* p;  
    p = AlocaRegDupla();  
    p->dado = -1;  
    p->esq = p;  
    p->dir = p;  
    return p;  
}
```

Listas duplamente ligadas

- Funções que percorrem a lista.

```
void ImprimeListaDuplaCircular(ListaDupla p) {
    RegDupla *q;
    q = p;
    do{
        ImprimeElemento(q->dado) ;
        q = q->dir;
    }while (q!=p) ;
    printf("\n");
}
```

```
void ImprimeListaDuplaCircularReversa(ListaDupla p) {
    RegDupla *q;
    q = p;
    do{
        ImprimeElemento(q->dado) ;
        q = q->esq;
    }while (q!=p) ;
    printf("\n");
}
```


Listas duplamente ligadas

- Funções de inserção e remoção com nó-cabeça.

```
/* Insere após o nó apontado por p. */  
void InsereDuplaCircular(ListaDupla p, TipoDado x) {  
    RegDupla *q;  
    q = AlocaRegDupla();  
    q->dado = x;  
    q->dir = p->dir;  
    q->esq = p;  
    (p->dir)->esq = q;  
    p->dir = q;  
}
```

```
/*Listas duplamente ligadas permitem a remoção do  
próprio nó passado como argumento.  
Cuidado para não remover nó-cabeça! */  
void RemoveDuplaCircular(ListaDupla p) {  
    (p->esq)->dir = p->dir;  
    (p->dir)->esq = p->esq;  
    free(p);  
}
```

Exemplos

- Manipulação de polinômios de grau $n \geq 0$.

- $$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0 x^0$$

- A solução a seguir usa listas circulares com nó-cabeça, a fim de utilizar a técnica de sentinelas. Supusemos:

- **a)** que cada nó representa um termo com coeficiente não nulo;

- **b)** Os termos estão em ordem decrescente dos expoentes;

- **c)** O nó-cabeça tem expoente -1 (conveniente para as operações).

Exemplos: polinômios

- Definição típica da estrutura utilizada.

```
typedef struct _Termo{  
    float        coef;  
    int          expo;  
    struct _Termo *prox;  
} Termo;
```

```
typedef Termo* Polinomio;
```

```
/*'Polinomio' é um tipo abstrato que esta  
sendo implementado. */
```

Exemplos: polinômios

- Criando polinômio nulo.

```
Termo* AlocaTermo() {  
    Termo* q;  
    q = (Termo*)calloc(1, sizeof(Termo));  
    if(q==NULL) exit(-1);  
    return q;  
}
```

```
Polinomio CriaPolinomioNulo() {  
    Termo* p;  
    p = AlocaTermo();  
    p->expo = -1;  
    p->coef = 0.0;  
    p->prox = p;  
    return p;  
}
```

Exemplos: polinômios

- Inserindo novo termo no polinômio.

```
void InserirTermo(Polinomio p,  
                 float coef,  
                 int expo) {  
    Termo *q, *aq, *x;  
  
    x = AlocaTermo();  
    x->coef = coef;  
    x->expo = expo;  
  
    aq = p;  
    q = p->prox;  
    while(q->expo > expo) {  
        aq = q;  
        q = q->prox;  
    }  
    x->prox = q;  
    aq->prox = x;  
}
```

Exemplos: polinômios

- Criando novo polinômio a partir de uma expressão (**string**).

```
/* Lê strings como: "5.0*x^4 + 7.5*x^2 + 3.0*x^1" */
Polinomio CriaPolinomio(char *expr) {
    Termo* p;
    float coef;
    int expo, r, n, nn=0;

    p = CriaPolinomioNulo();
    do{
        r = sscanf(expr+nn, " %f * x ^ %d%n", &coef, &expo, &n);
        if(r==EOF || r==0) break;
        nn += n;

        InserirTermo(p, coef, expo);

        r = sscanf(expr+nn, " + %n", &n);
        if(r==EOF) break;
        nn += n;
    }while(1);
    return p;
}
```

Exemplos: polinômios

- Exemplo de função que imprime o polinômio.

```
void ImprimeTermo(Termo *q) {
    printf("%f*x^%d", q->coef, q->expo);
}

void ImprimePolinomio(Polinomio p) {
    Termo *q;

    q = p->prox;
    if(q==p) return;

    ImprimeTermo(q);
    q = q->prox;

    while(q!=p) {
        printf(" + ");
        ImprimeTermo(q);
        q = q->prox;
    }
    printf("\n");
}
```

Exemplos: polinômios

- Exemplo de função que soma dois polinômios:
 - Ela percorre simultaneamente as duas listas,
 - somando os termos com expoentes iguais e
 - copiando de modo intercalado aqueles que não tem correspondente no outro.

Exemplos: polinômios

```
Polinomio SomaPolinomios(Polinomio p,
                          Polinomio q) {
    Polinomio r;
    Termo *pp, *qq, *rr;
    int ep, eq;
    float cf;

    r = CriaPolinomioNulo();
    pp = p->prox;
    qq = q->prox;
    rr = r;
    do{
        ep = pp->expo;
        eq = qq->expo;
        if(ep>eq) {
            InereTermo(rr, pp->coef, ep);
            pp = pp->prox;
            rr = rr->prox;
        }
        else if(ep<eq) {
            InereTermo(rr, qq->coef, eq);
            qq = qq->prox;
            rr = rr->prox;
        }
    }
```

```
        else if(ep>-1) { /* eq==ep */
            cf = pp->coef + qq->coef;
            if(cf!=0.0) {
                InereTermo(rr, cf, ep);
                rr = rr->prox;
            }
            pp = pp->prox;
            qq = qq->prox;
        }
    }while((ep>-1) || (eq>-1));

    return r;
}
```