Estrutura de Dados

Hamilton José Brumatto

Bacharelado em Ciências da Computação - UESC

15 de abril de 2016

Listas Encadeadas

Sequência dinâmica de elementos, normalmente do mesmo tipo

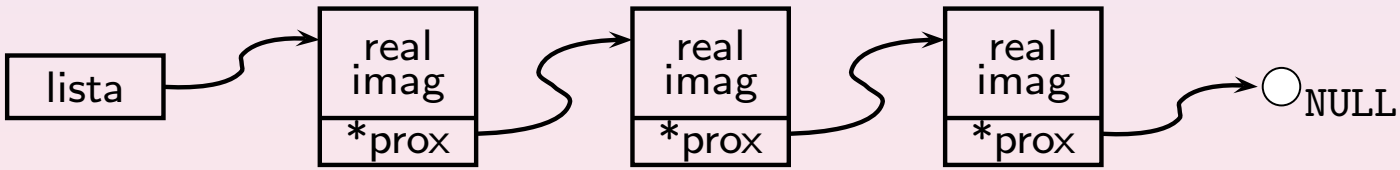
- Uma lista encadeada (ou ligada) permite criar um modelo dinâmico de sequência.
- Não há uma indexação dos elementos, um elemento está ligado ao próximo.
- O acesso não é aleatório, e sim sequencial.
  - Para chegar em uma posição tem de passar pelos anteriores.
- A navegação entre os elementos se dá através de ponteiros.
- A ligação entre os elementos se dá através de ponteiros.
- Cada elemento é criado dinamicamente através de alocação de memória.
- Um ponteiro especial: "cabeça", sempre aponta para o primeiro elemento da lista.

Exemplo básico de uma lista encadeada

- Declarando a estrutura

Estrutura com ponteiro para o próximo

```
typedef struct st_complexo {
    double real;
    double imag;
    struct st_complexo *prox; } complexo;
```



`NULL` *é uma constante declarada em alguns includes, por exemplo,* `stdlib.h`

Manipulando a estrutura

Programa principal

```
int main(int argc, char **args) {
      complexo *lista, *p;
      lista = malloc(sizeof(complexo));
      p = lista;
      p->real = 2; p->imag = 3;
      p->prox = malloc(sizeof(complexo));
      p = p->prox;
      p->real = 3; p->imag = 4;
      p->prox = malloc(sizeof(complexo));
      p = p->prox;
      p->real = 4; p->imag = 5; p->prox = NULL;
      p = lista;
      while(p != NULL) {
            printf("(%f + i%f)..",p->real,p->imag); p=p->prox;
      }
      printf("\n");
      return 0;
}
```

Saída do exemplo acima

```
(2.000000 + i3.000000)..(3.000000 + i4.000000)..(4.000000 + i5.000000)..
```

Ponteiro para ponteiro para ...

Apontando:

```
p = lista;
printf("(%f + i%f)\n",p->prox->real, p->prox->prox->imag);
```

Resultado:

```
(3.000000 + i5.000000)
```

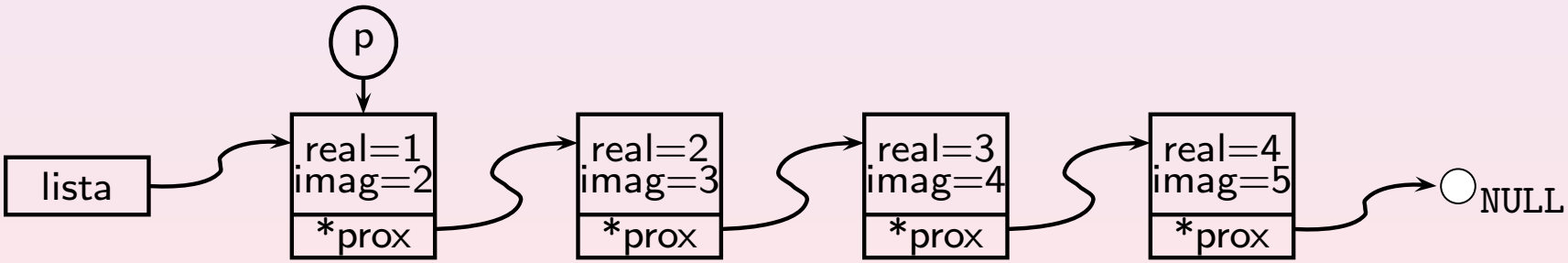
Manipulando listas encadeadas:

- Percorrendo uma lista encadeada
- Inserindo elementos
- Removendo elementos
- Trocando elementos de posição

Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

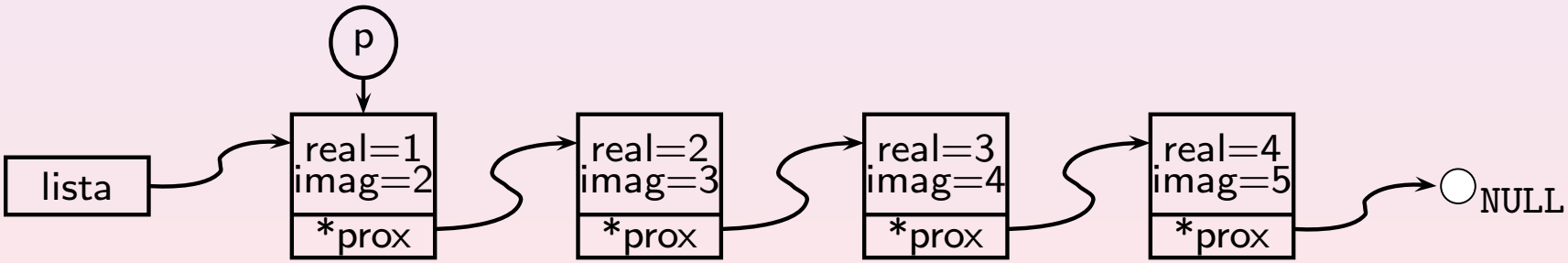
Resultados



Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

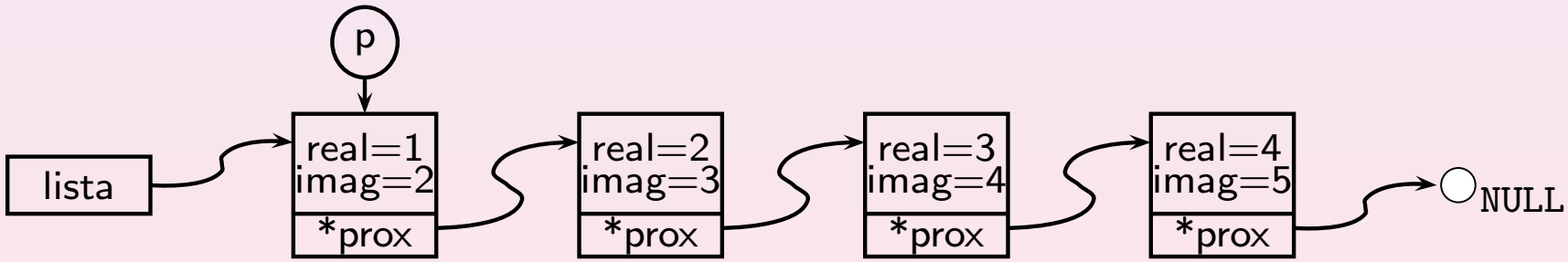


Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

```
(1.000000 + i2.000000)
```

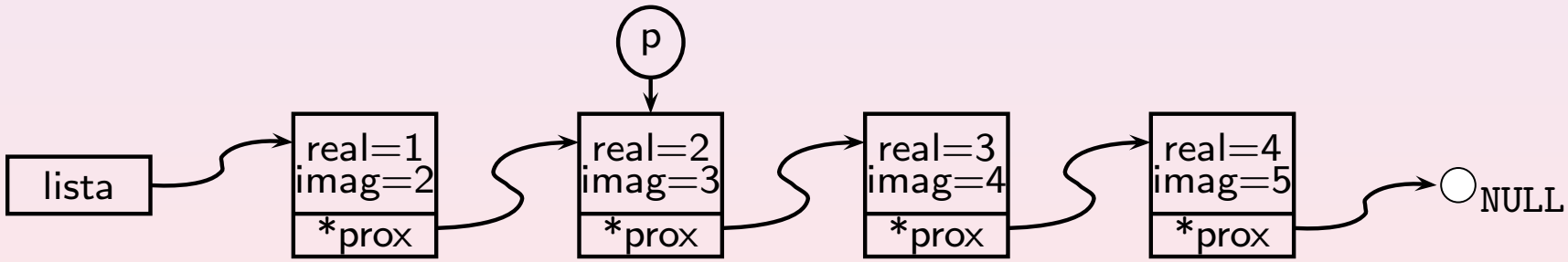


Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

```
(1.000000 + i2.000000)
```

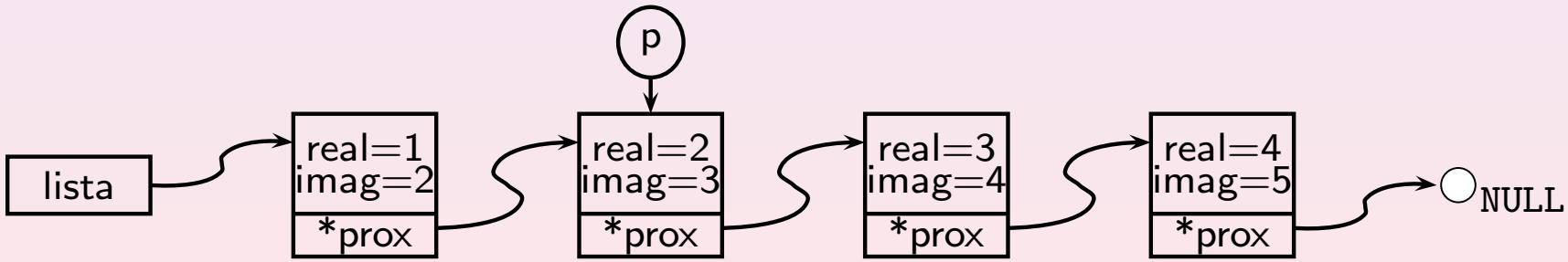


Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

```
(1.000000 + i2.000000)
```

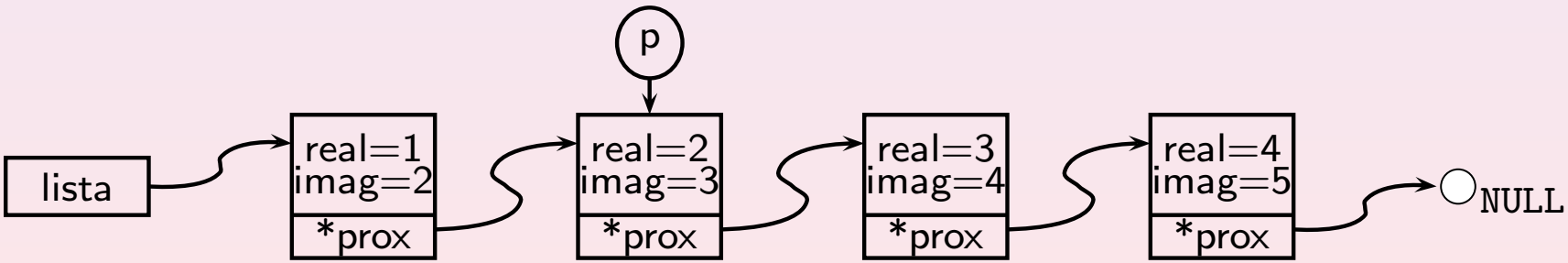


Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
```

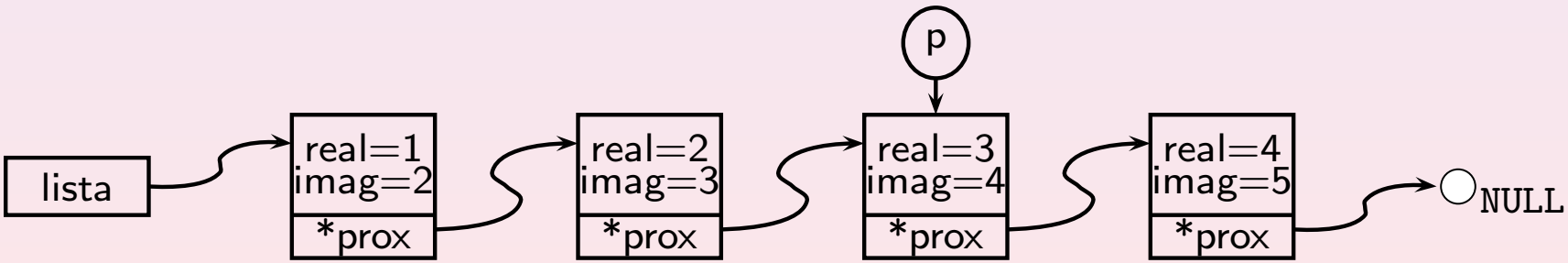


Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
```

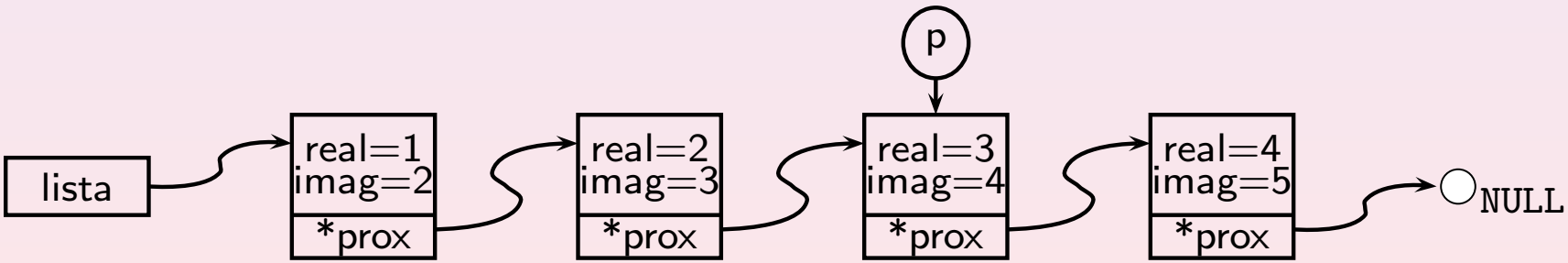


Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
```

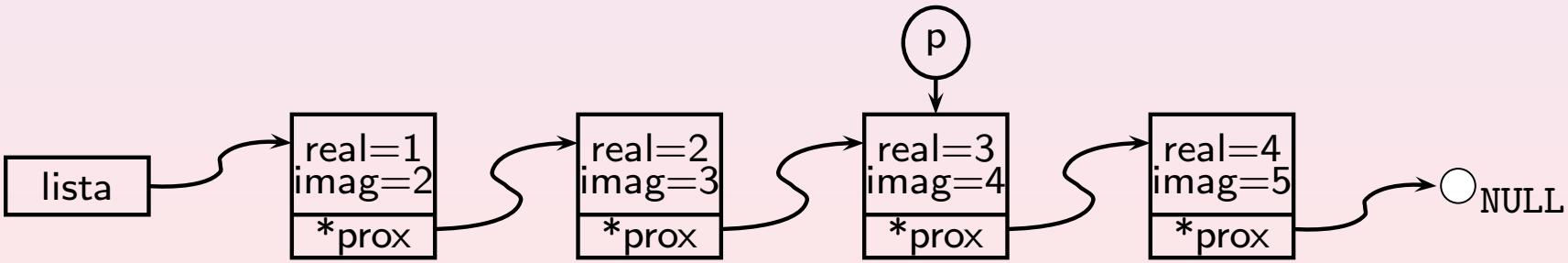


Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(3.000000 + i4.000000)
```

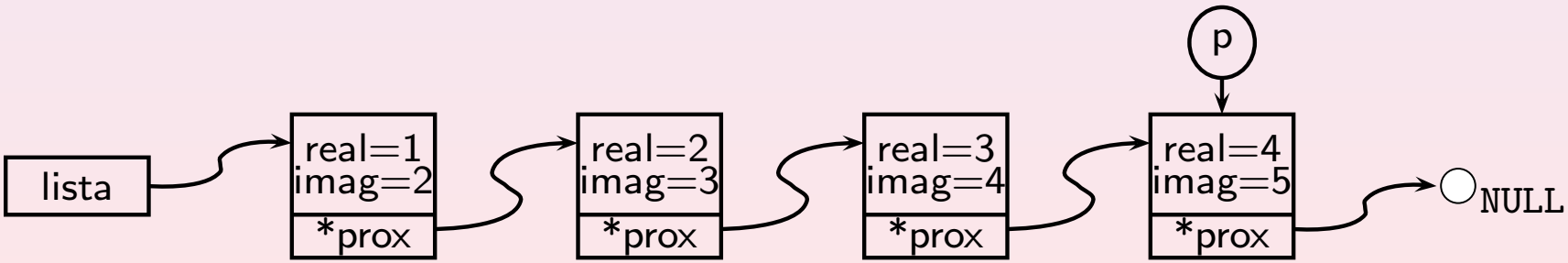


Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(3.000000 + i4.000000)
```

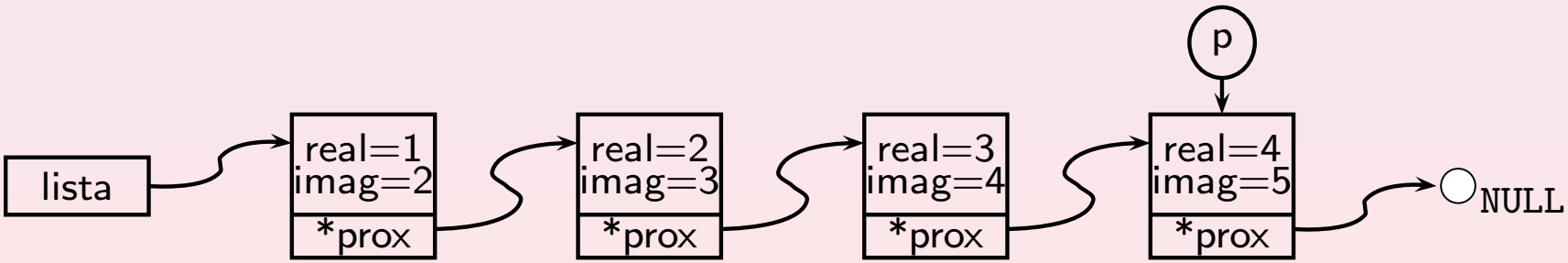


Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(3.000000 + i4.000000)
```

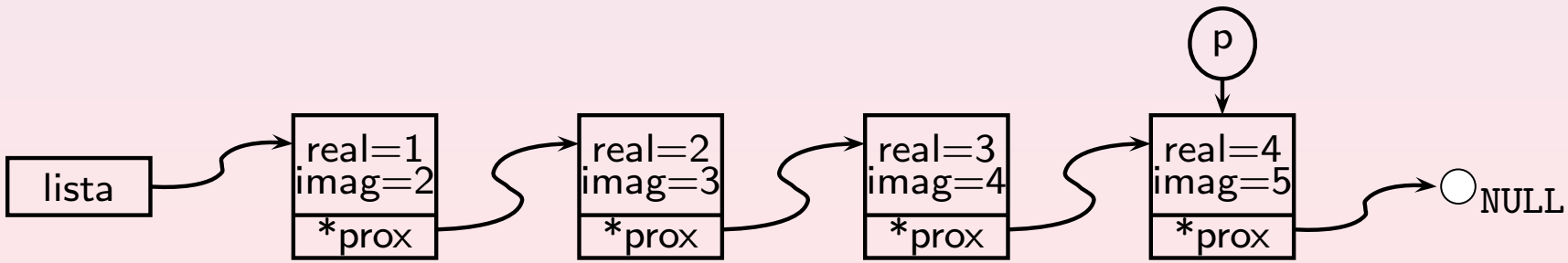


Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(3.000000 + i4.000000)
(4.000000 + i5.000000)
```

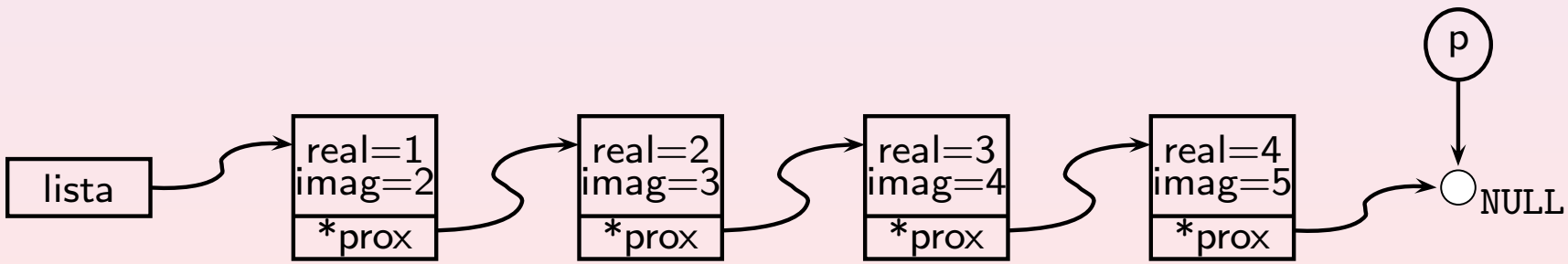


Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(3.000000 + i4.000000)
(4.000000 + i5.000000)
```

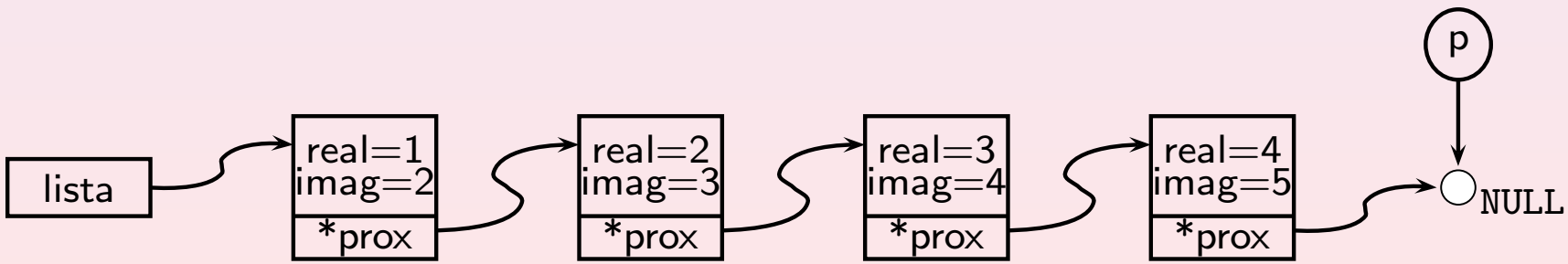


Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(3.000000 + i4.000000)
(4.000000 + i5.000000)
```

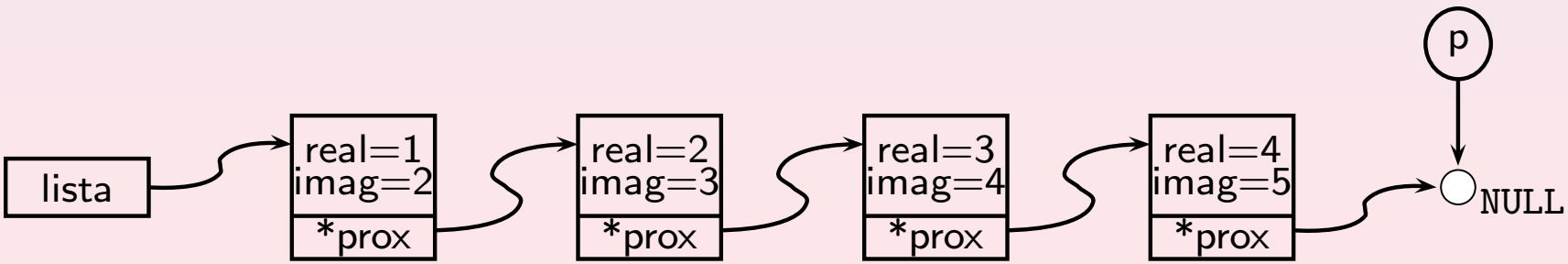


Navegando por ponteiros

```
p = lista;
while(p != NULL) {
    printf("(%f + i%f)\n",p->real,p->imag);
    p=p->prox;
}
```

Resultados

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(3.000000 + i4.000000)
(4.000000 + i5.000000)
```

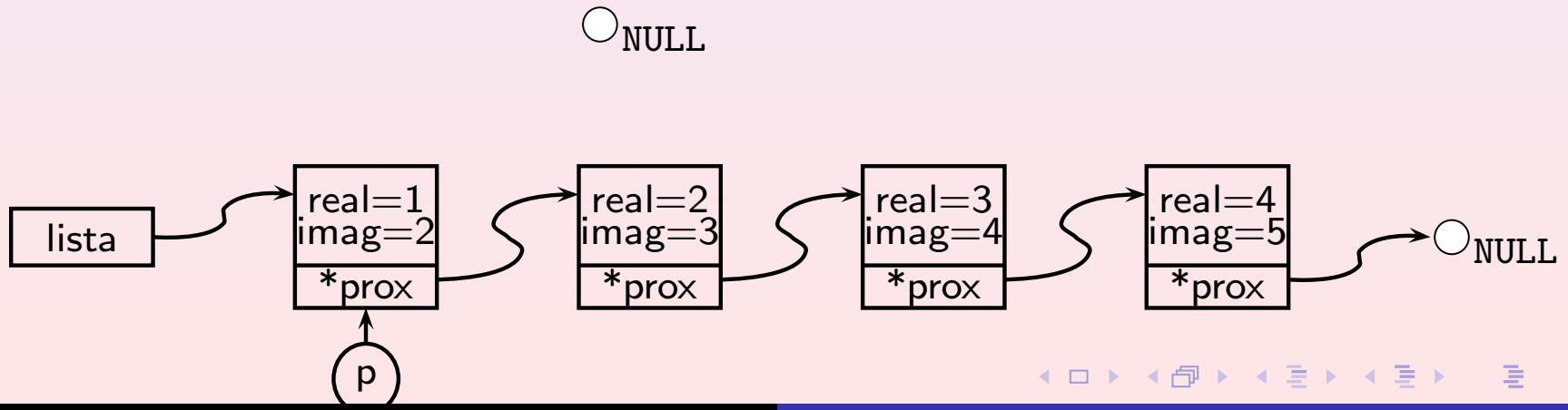


Inserindo elemento no início da lista

```
n = malloc(sizeof(complexo));
n->real = 5; n->imag = 6;
n->prox = NULL;
n->prox = p;
if(p == lista) {
    lista = n;
}
```

Resultado

```
(5.000000 + i6.000000)
(1.000000 + i2.000000)...
```

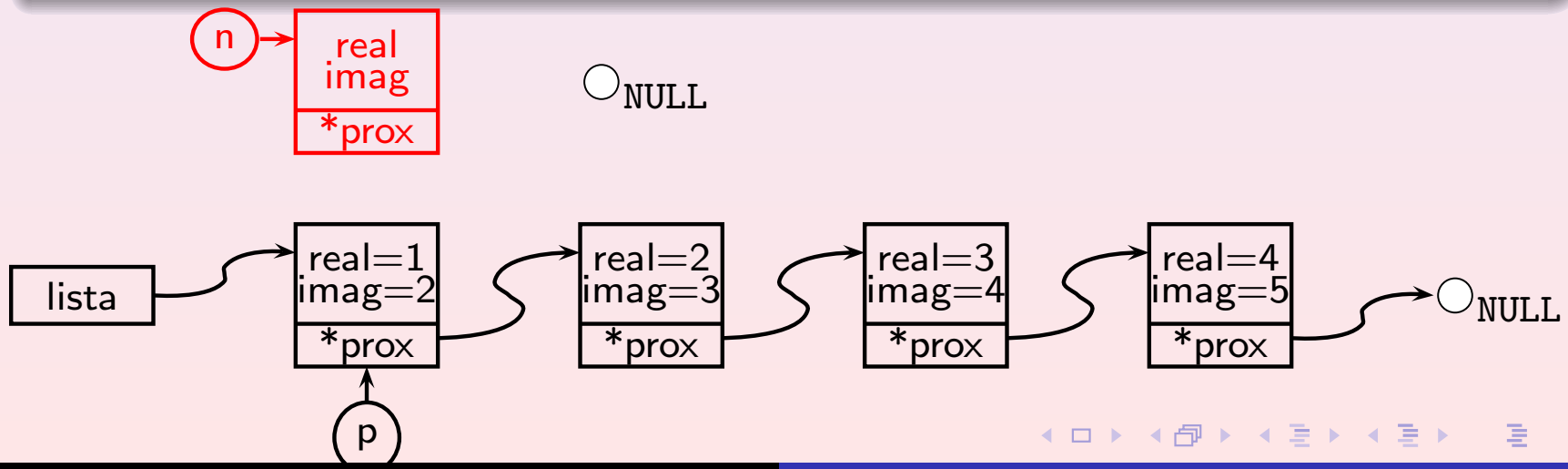


Inserindo elemento no início da lista

```
n = malloc(sizeof(complexo));
n->real = 5; n->imag = 6;
n->prox = NULL;
n->prox = p;
if(p == lista) {
    lista = n;
}
```

Resultado

```
(5.000000 + i6.000000)
(1.000000 + i2.000000)...
```

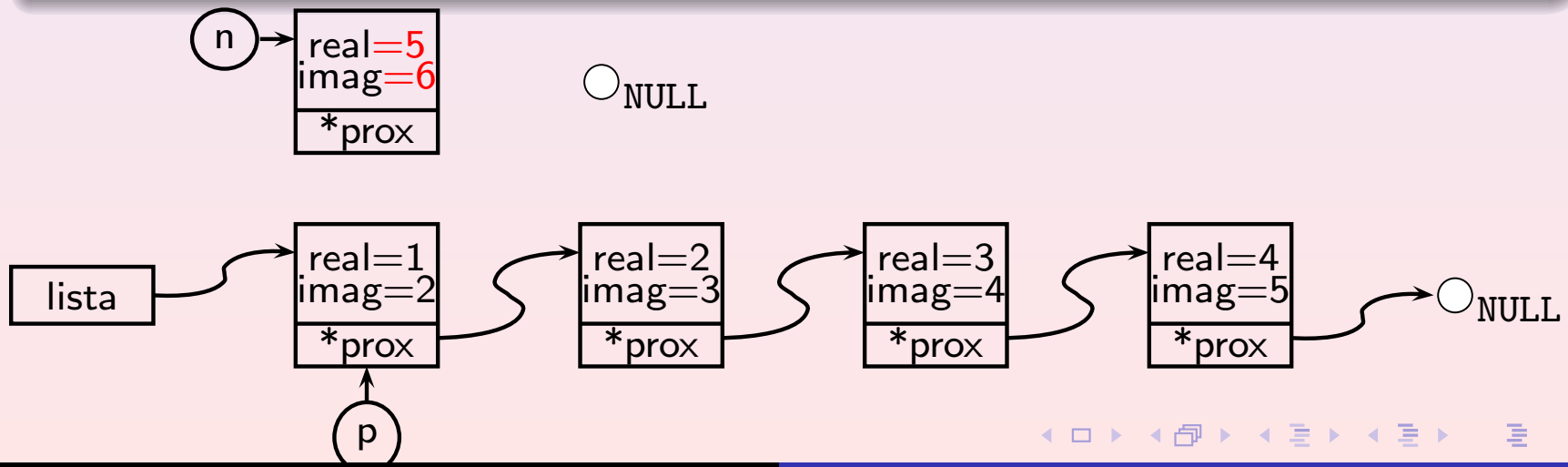


Inserindo elemento no início da lista

```
n = malloc(sizeof(complexo));
n->real = 5; n->imag = 6;
n->prox = NULL;
n->prox = p;
if(p == lista) {
    lista = n;
}
```

Resultado

```
(5.000000 + i6.000000)
(1.000000 + i2.000000)...
```

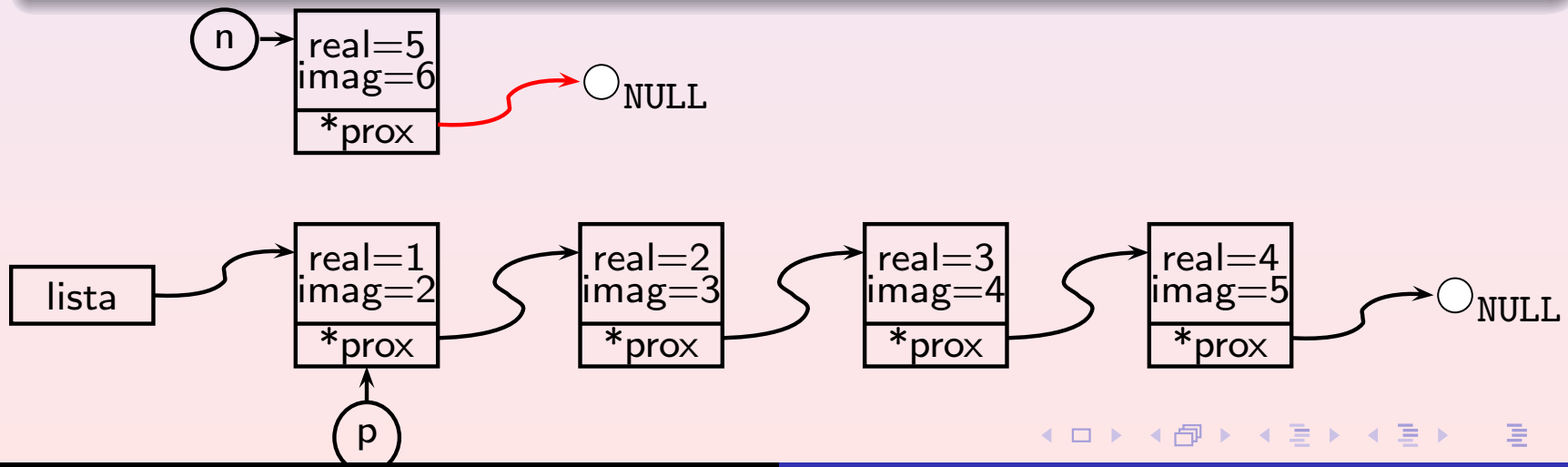


Inserindo elemento no início da lista

```
n = malloc(sizeof(complexo));
n->real = 5; n->imag = 6;
n->prox = NULL;
n->prox = p;
if(p == lista) {
    lista = n;
}
```

Resultado

```
(5.000000 + i6.000000)
(1.000000 + i2.000000)...
```

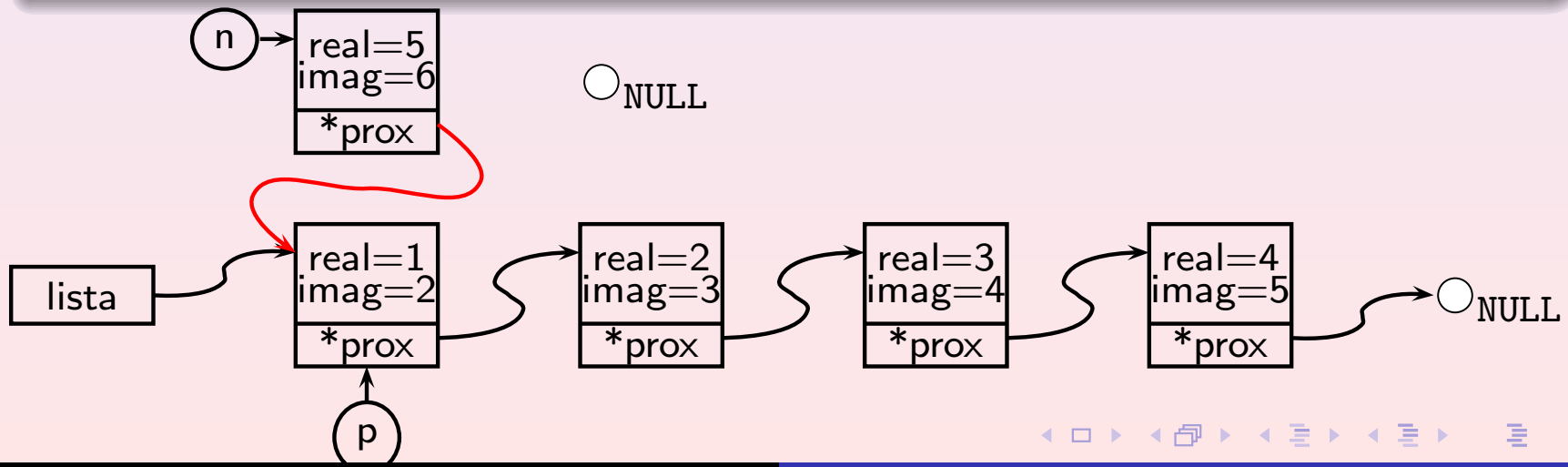


Inserindo elemento no início da lista

```
n = malloc(sizeof(complexo));
n->real = 5; n->imag = 6;
n->prox = NULL;
n->prox = p;
if(p == lista) {
    lista = n;
}
```

Resultado

```
(5.000000 + i6.000000)
(1.000000 + i2.000000)...
```

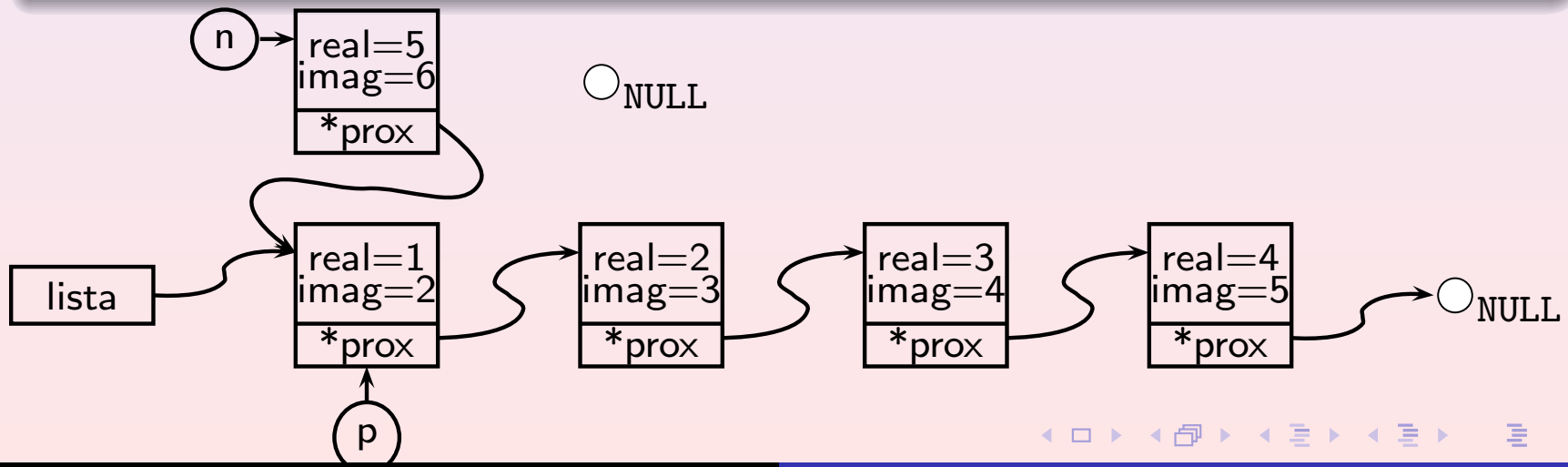


Inserindo elemento no início da lista

```
n = malloc(sizeof(complexo));
n->real = 5; n->imag = 6;
n->prox = NULL;
n->prox = p;
if(p == lista) {
    lista = n;
}
```

Resultado

```
(5.000000 + i6.000000)
(1.000000 + i2.000000)...
```

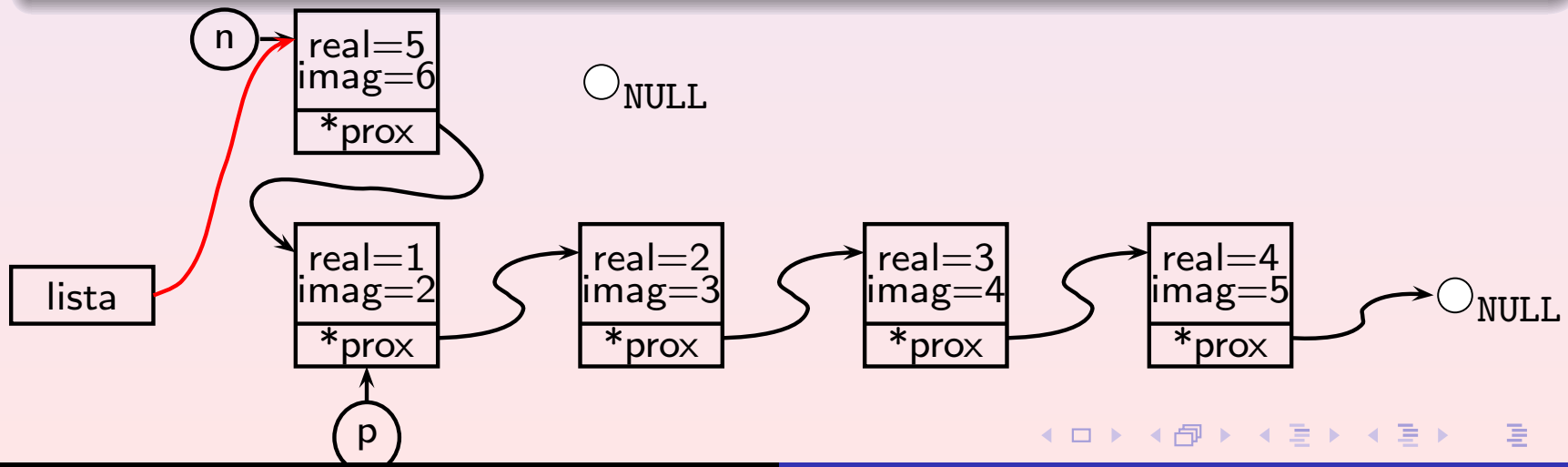


Inserindo elemento no início da lista

```
n = malloc(sizeof(complexo));
n->real = 5; n->imag = 6;
n->prox = NULL;
n->prox = p;
if(p == lista) {
    lista = n;
}
```

Resultado

```
(5.000000 + i6.000000)
(1.000000 + i2.000000)...
```

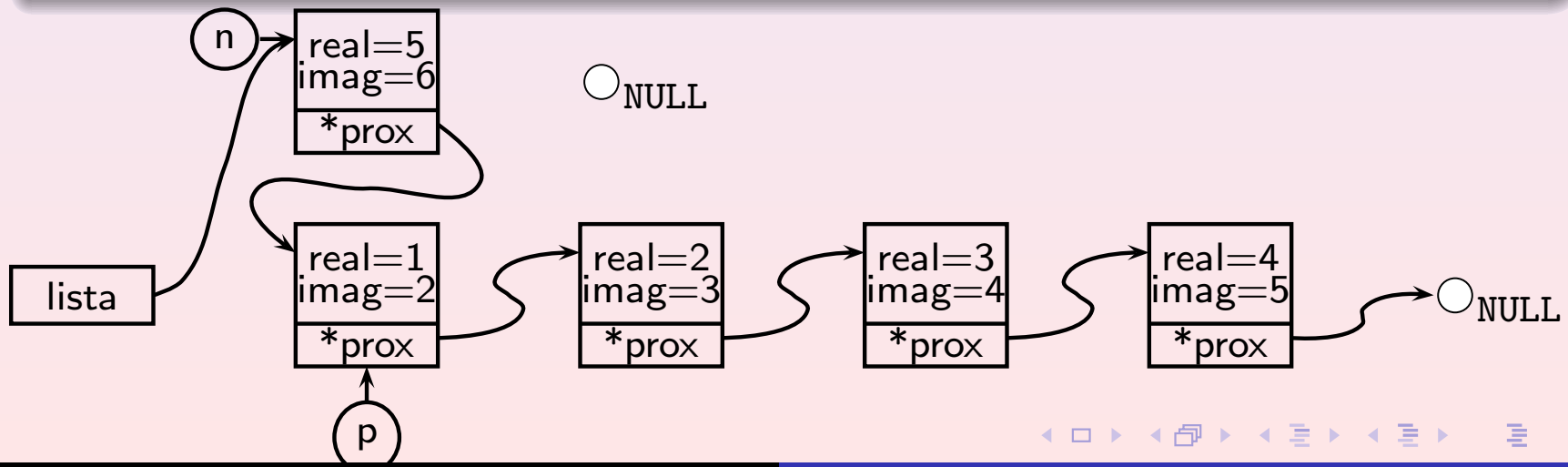


Inserindo elemento no início da lista

```
n = malloc(sizeof(complexo));
n->real = 5; n->imag = 6;
n->prox = NULL;
n->prox = p;
if(p == lista) {
    lista = n;
}
```

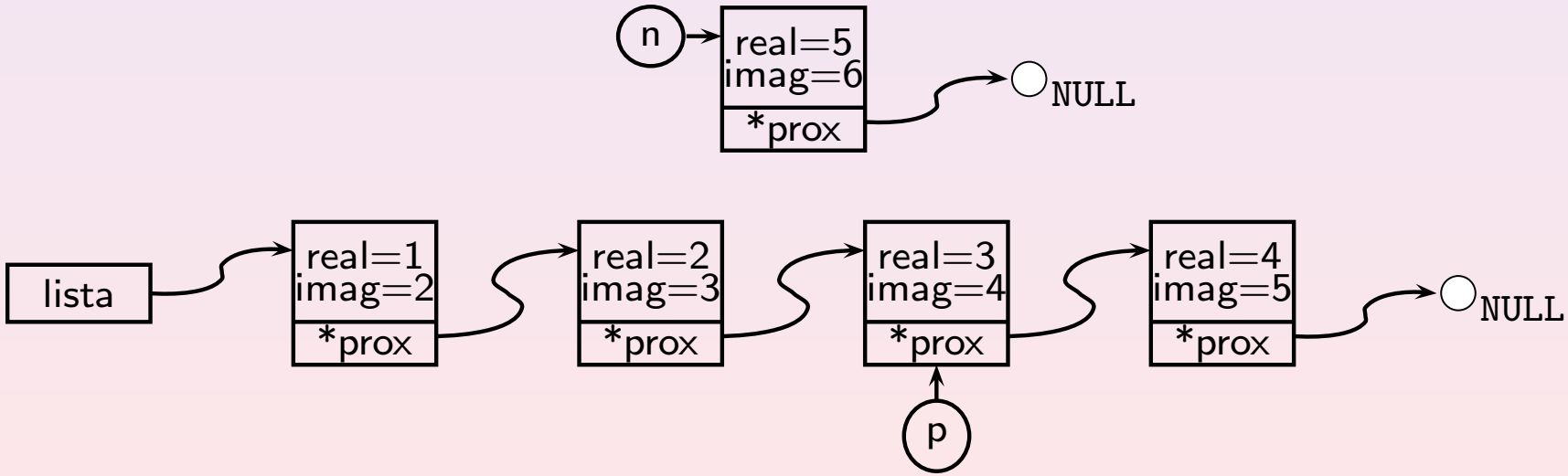
Resultado

```
(5.000000 + i6.000000)
(1.000000 + i2.000000)...
```



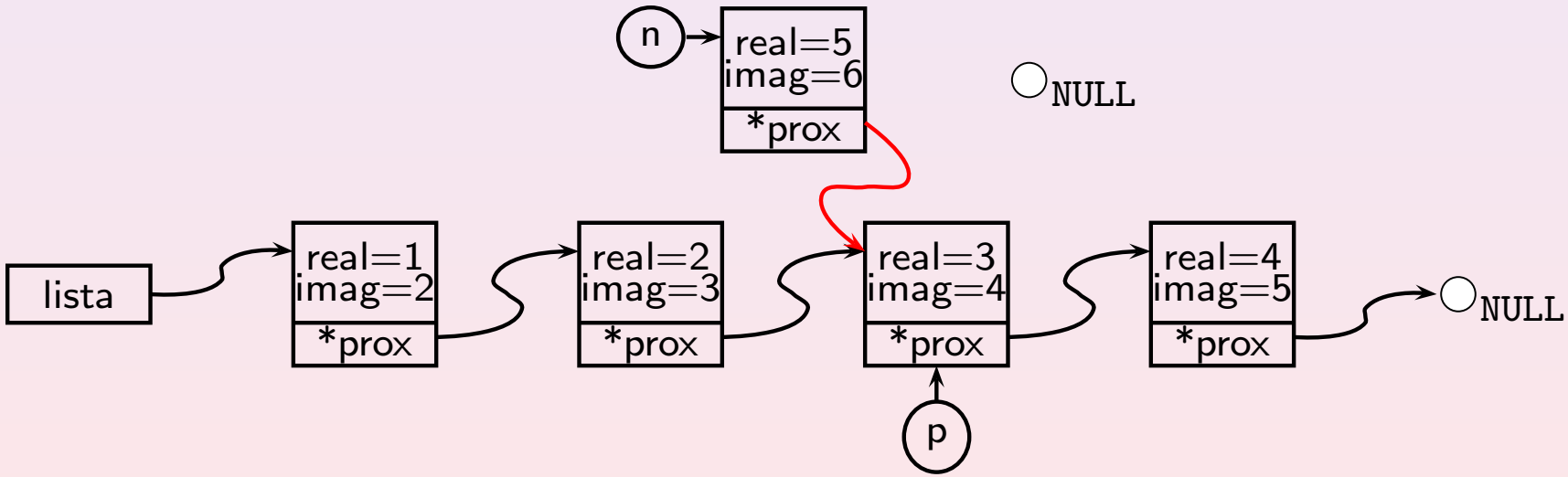
Inserindo elemento do meio pro fim da lista

```
n->prox = p;
if(p == lista) {
    lista = n;
} else {
    a = lista;
    while(a->prox != p) a = a->prox;
    a->prox = n;
}
```



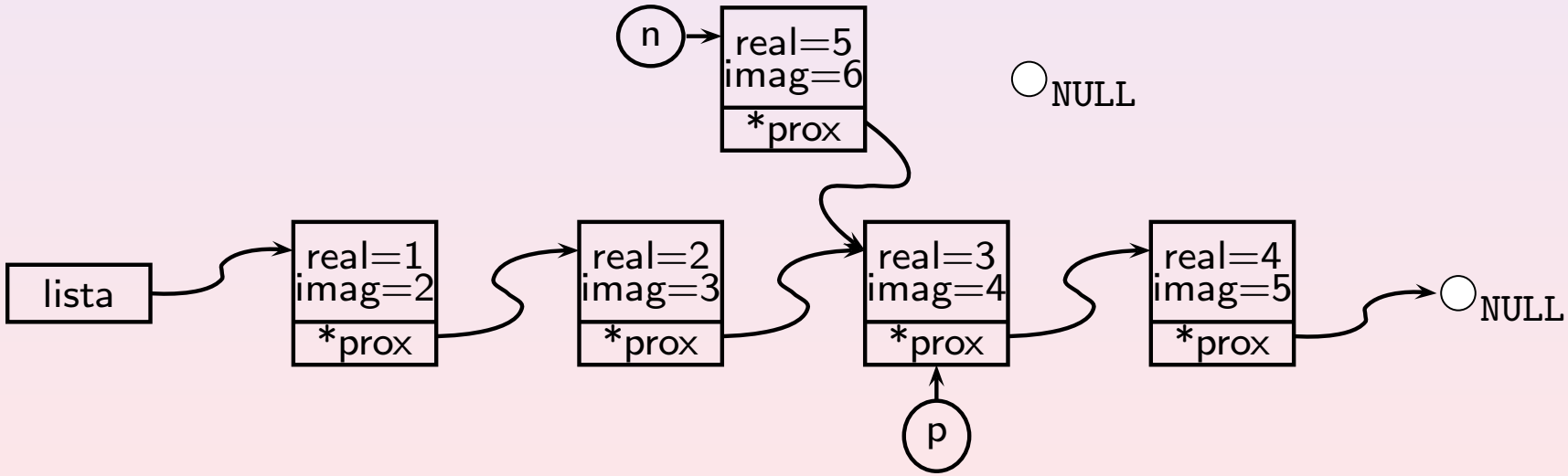
Inserindo elemento do meio pro fim da lista

```
n->prox = p;
if(p == lista) {
    lista = n;
    a = lista;
    while(a->prox != p) a = a->prox;
    a->prox = n;
}
```



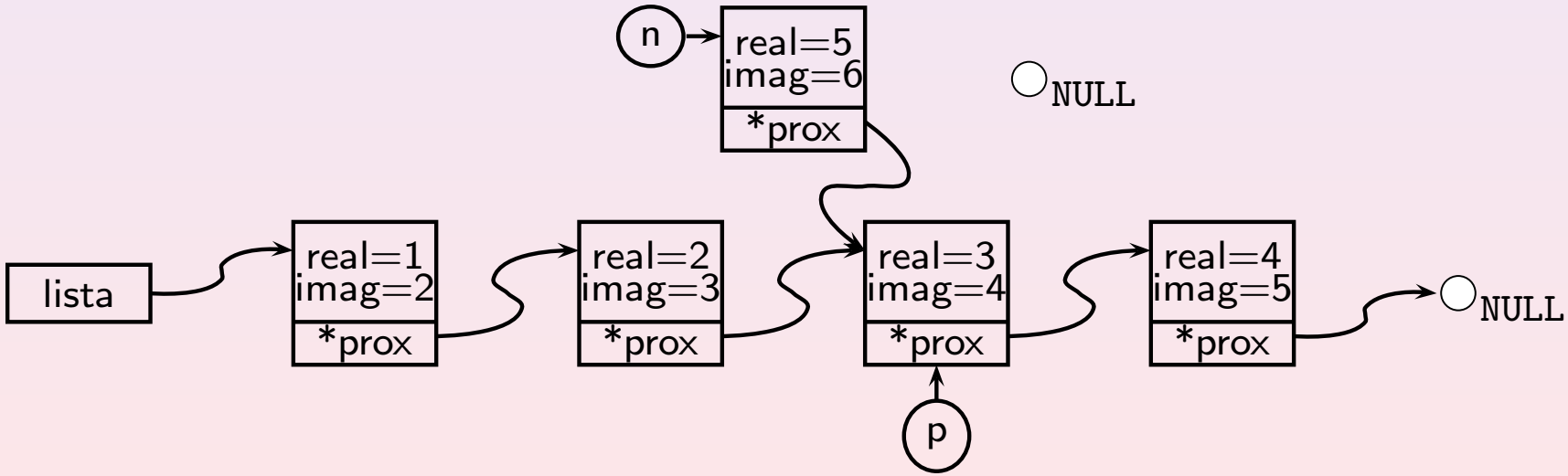
Inserindo elemento do meio pro fim da lista

```
n->prox = p;
if(p == lista) {
    lista = n;
} else {
    a = lista;
    while(a->prox != p) a = a->prox;
    a->prox = n;
}
```



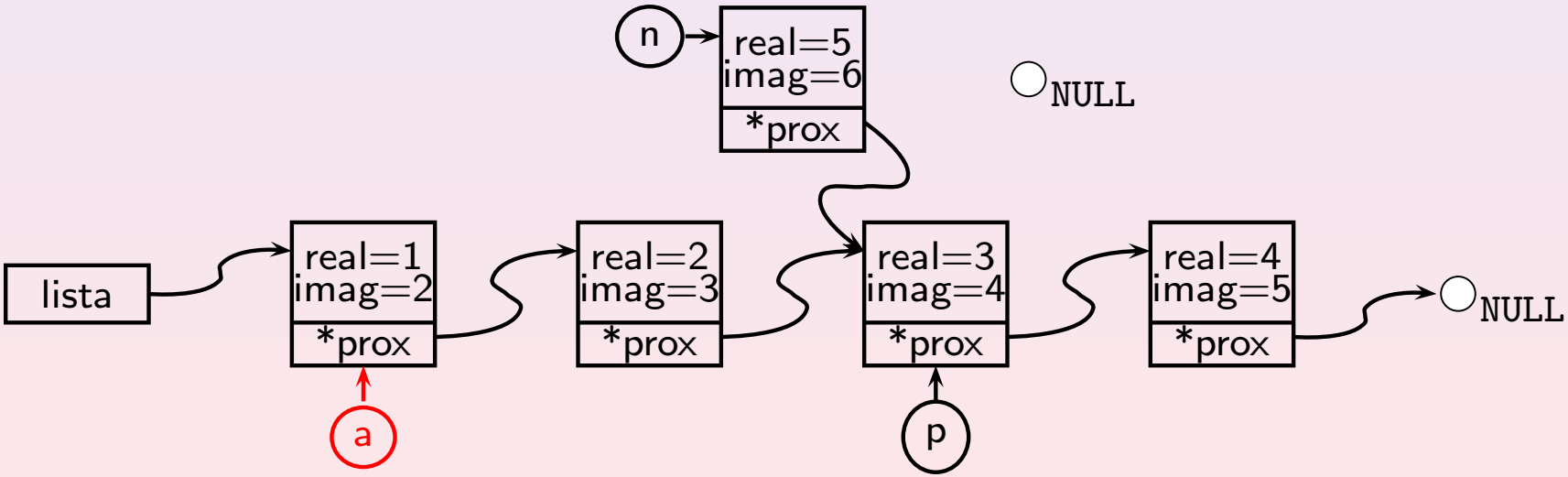
Inserindo elemento do meio pro fim da lista

```
n->prox = p;
if(p == lista) {
    lista = n;
} else {
    a = lista;
    while(a->prox != p) a = a->prox;
    a->prox = n;
}
```



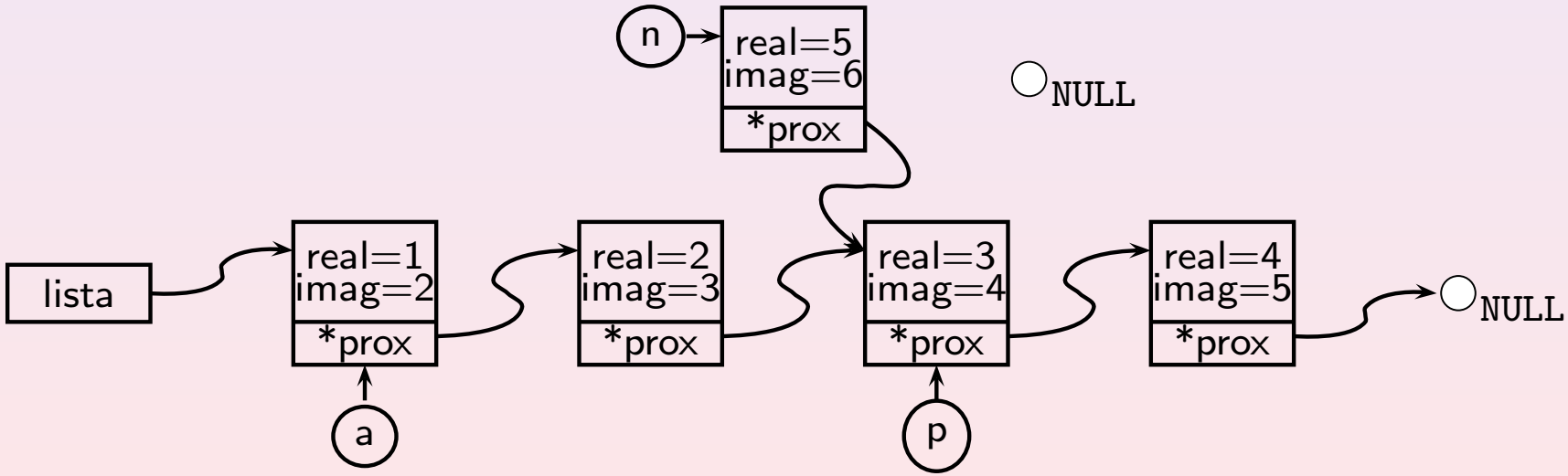
Inserindo elemento do meio pro fim da lista

```
n->prox = p;
if(p == lista) {
    lista = n;
} else {
    a = lista;
    while(a->prox != p) a = a->prox;
    a->prox = n;
}
```



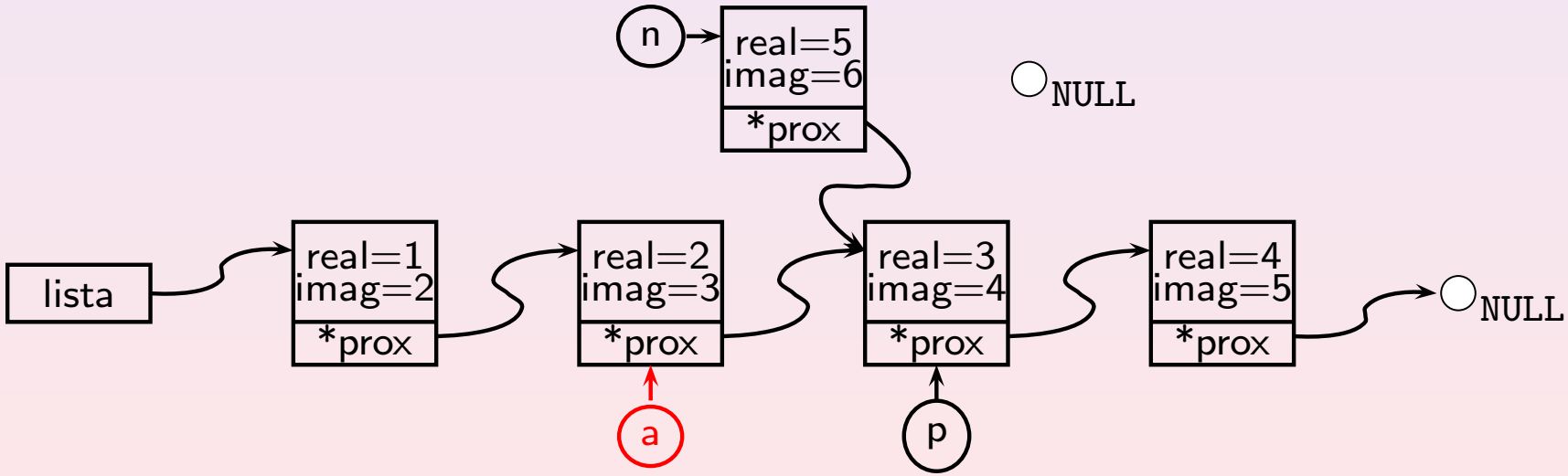
Inserindo elemento do meio pro fim da lista

```
n->prox = p;
if(p == lista) {
    lista = n;
} else {
    a = lista;
    while(a->prox != p) a = a->prox;
    a->prox = n;
}
```



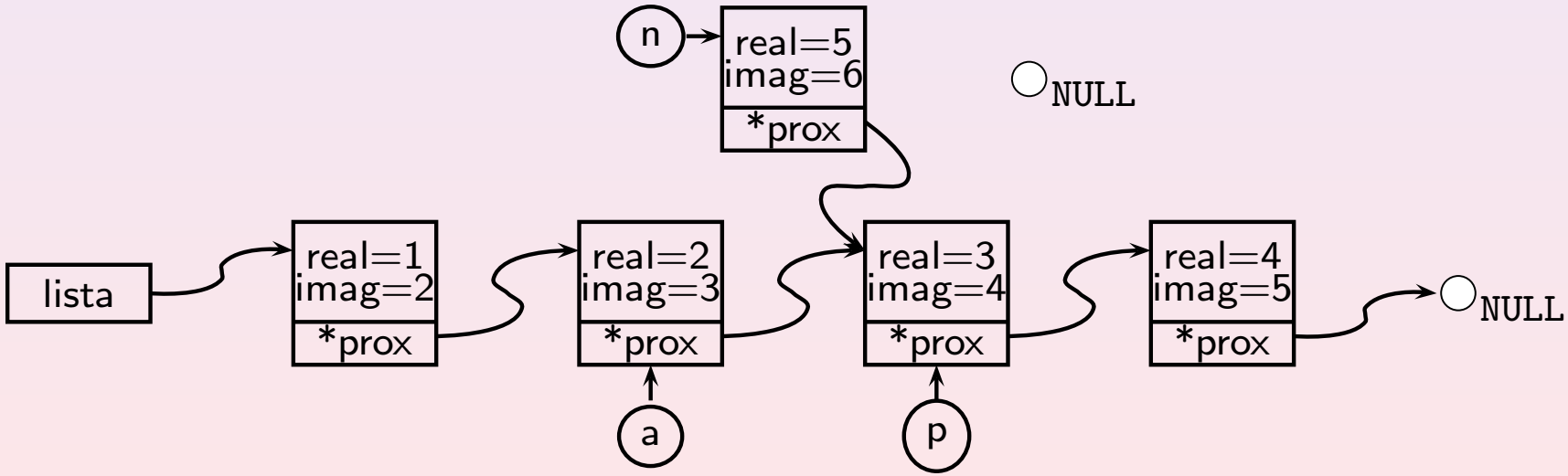
Inserindo elemento do meio pro fim da lista

```
n->prox = p;
if(p == lista) {
    lista = n;
} else {
    a = lista;
    while(a->prox != p) a = a->prox;
    a->prox = n;
}
```



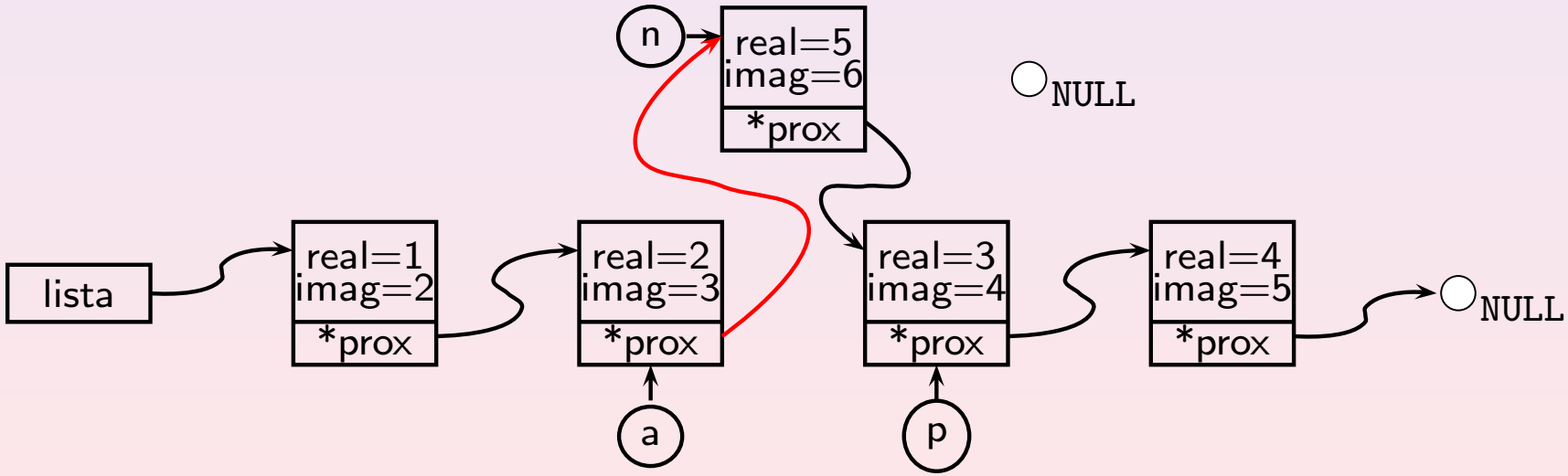
Inserindo elemento do meio pro fim da lista

```
n->prox = p;
if(p == lista) {
    lista = n;
} else {
    a = lista;
    while(a->prox != p) a = a->prox;
    a->prox = n;
}
```



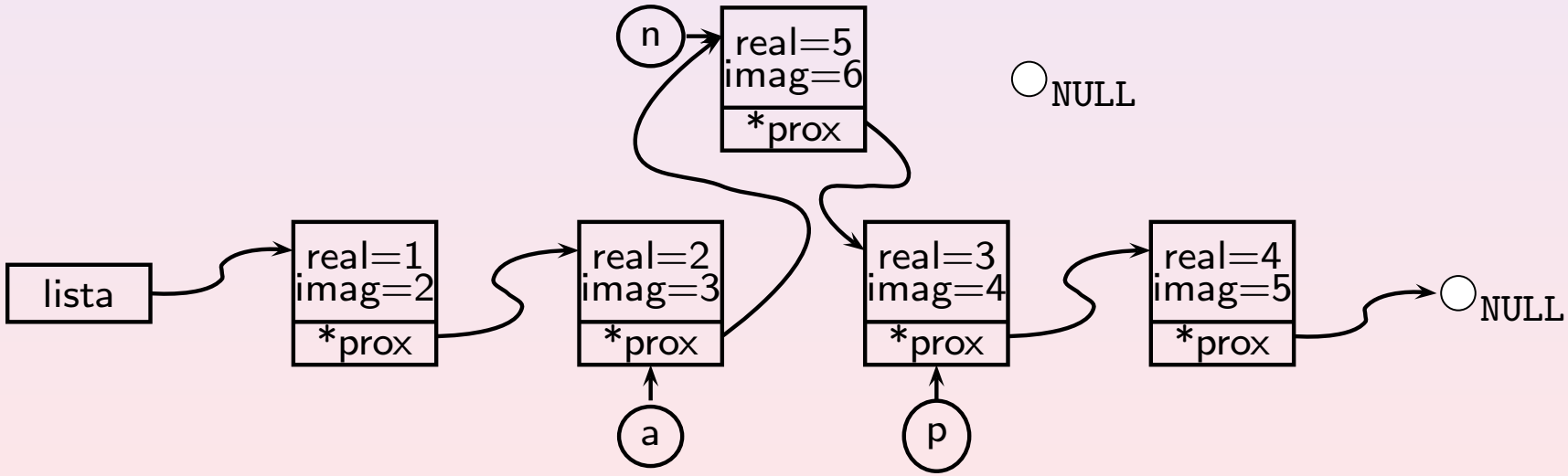
Inserindo elemento do meio pro fim da lista

```
n->prox = p;
if(p == lista) {
    lista = n;
} else {
    a = lista;
    while(a->prox != p) a = a->prox;
    a->prox = n;
}
```



Inserindo elemento do meio pro fim da lista

```
n->prox = p;
if(p == lista) {
    lista = n;
} else {
    a = lista;
    while(a->prox != p) a = a->prox;
    a->prox = n;
}
```

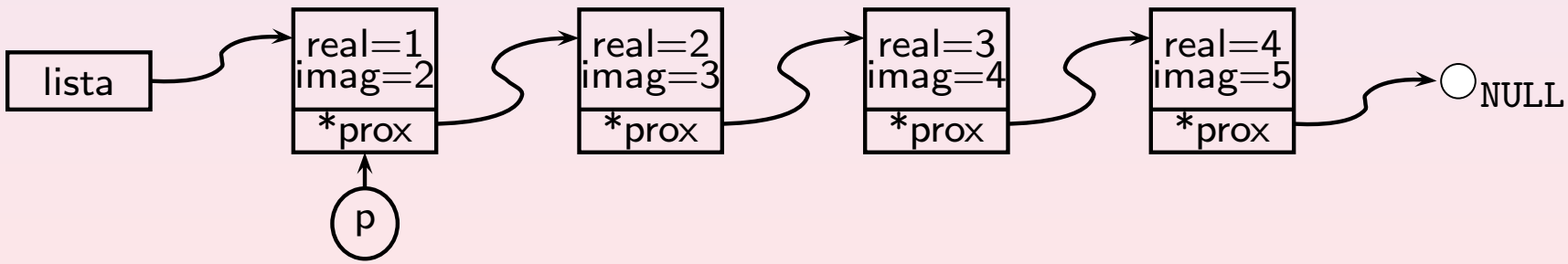


Removendo o primeiro elemento da lista

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

Resultado

```
(2.000000 + i3.000000)
(3.000000 + i4.000000)
(4.000000 + i5.000000)
```

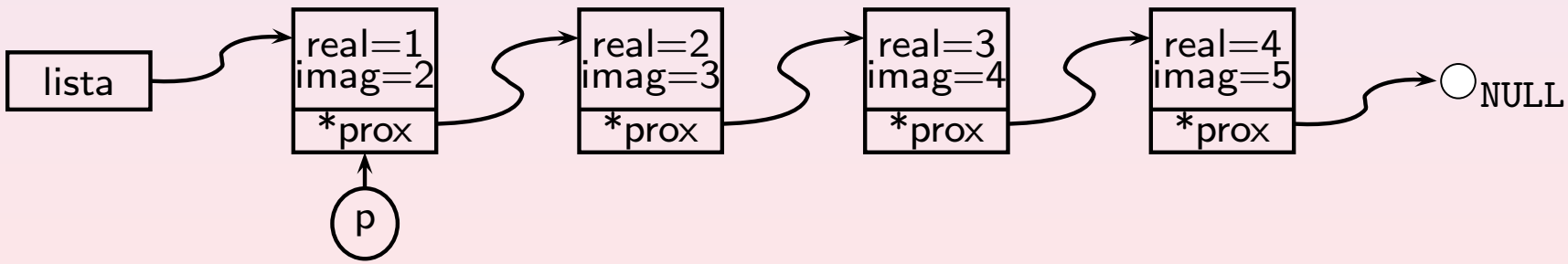


Removendo o primeiro elemento da lista

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

Resultado

```
(2.000000 + i3.000000)
(3.000000 + i4.000000)
(4.000000 + i5.000000)
```

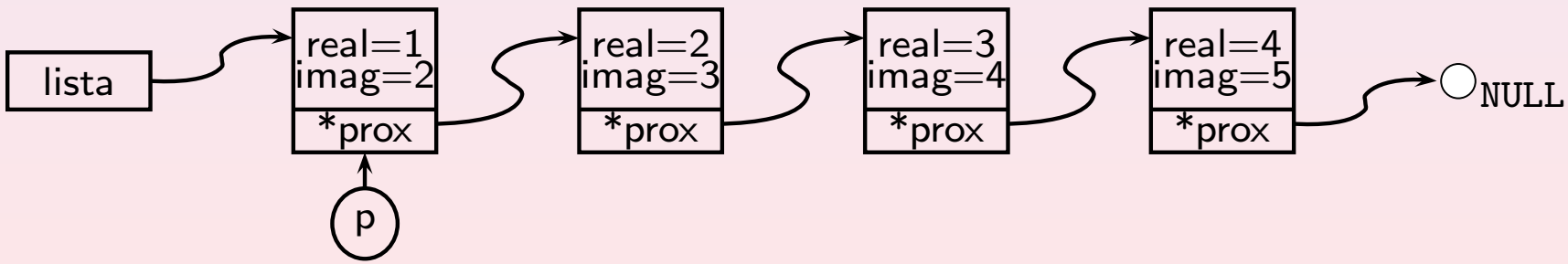


Removendo o primeiro elemento da lista

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

Resultado

```
(2.000000 + i3.000000)
(3.000000 + i4.000000)
(4.000000 + i5.000000)
```

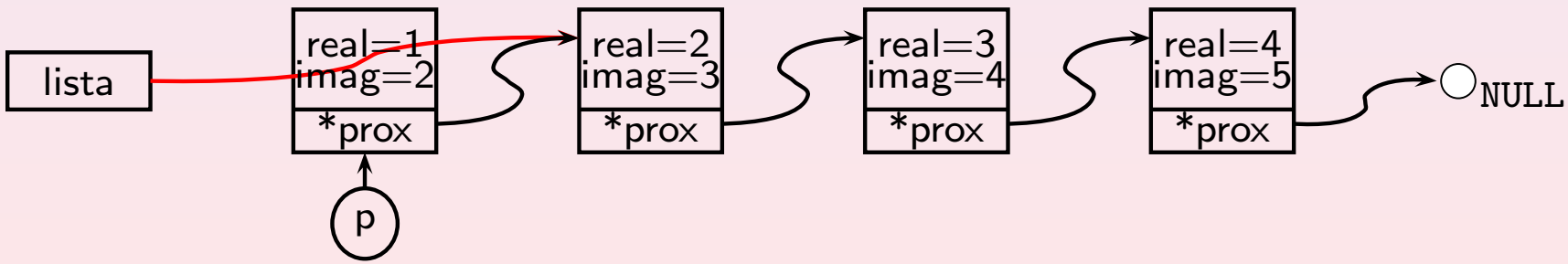


Removendo o primeiro elemento da lista

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

Resultado

```
(2.000000 + i3.000000)
(3.000000 + i4.000000)
(4.000000 + i5.000000)
```

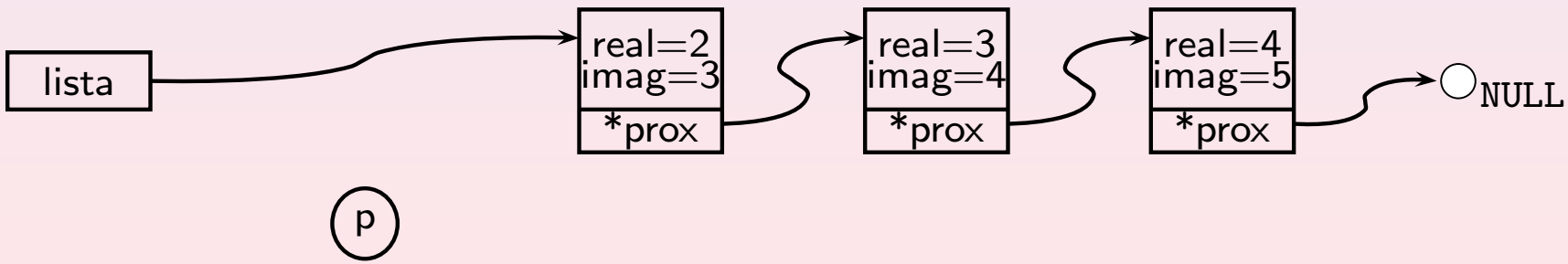


Removendo o primeiro elemento da lista

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

Resultado

```
(2.000000 + i3.000000)
(3.000000 + i4.000000)
(4.000000 + i5.000000)
```

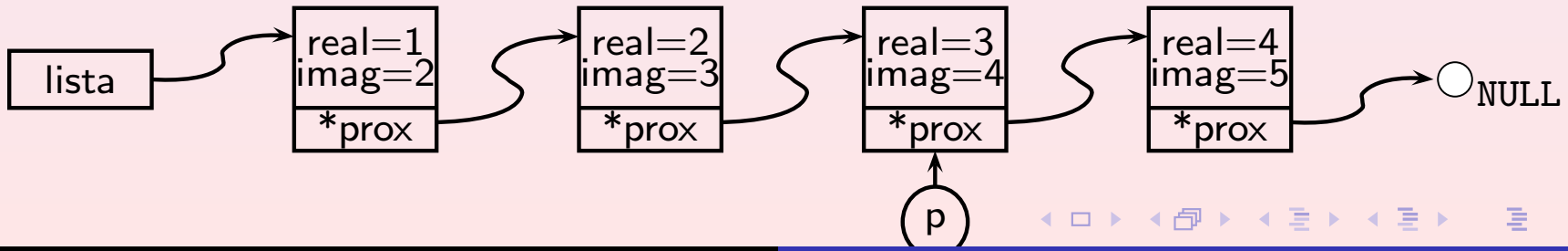


Removendo um elemento do meio pro fim

```
if(p != NULL) {
    if(p == lista) {
        lista = p->prox;
    } else {
        a = lista;
        while(a->prox != p)  a = a->prox;
        a->prox = p->prox;
    }
} free(p);
```

Resultado

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(4.000000 + i5.000000)
```

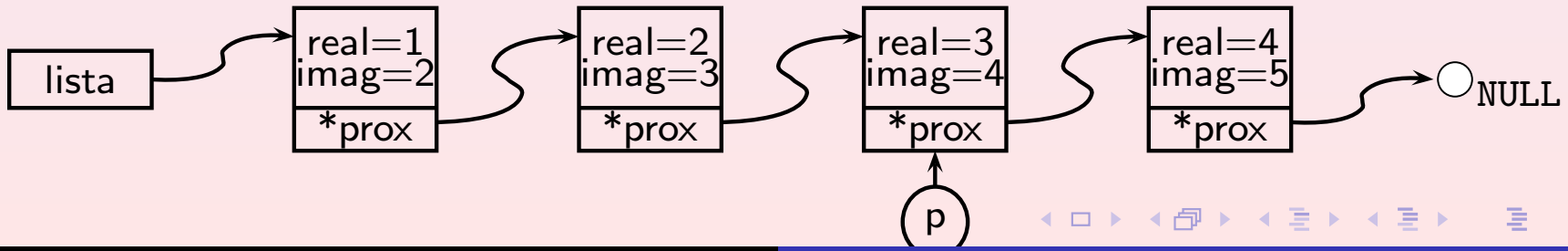


Removendo um elemento do meio pro fim

```
if(p != NULL) {
    if(p == lista) {
        lista = p->prox;
    } else {
        a = lista;
        while(a->prox != p)  a = a->prox;
        a->prox = p->prox;
    }
} free(p);
```

Resultado

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(4.000000 + i5.000000)
```

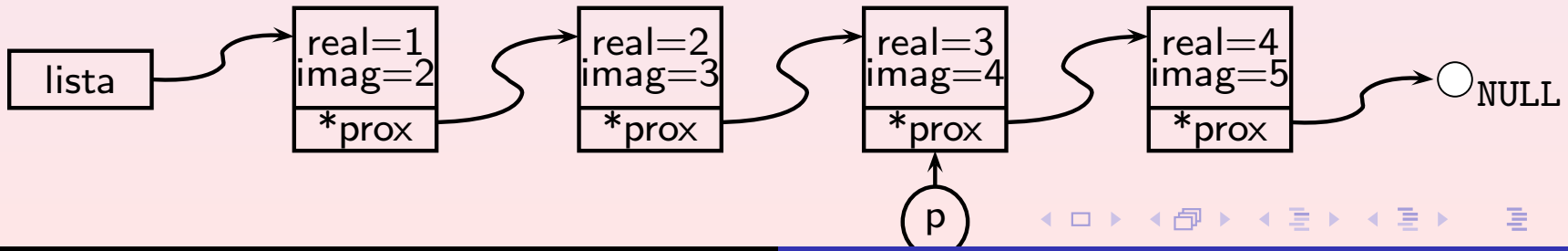


Removendo um elemento do meio pro fim

```
if(p != NULL) {
    if(p == lista) {
        lista = p->prox;
    } else {
        a = lista;
        while(a->prox != p)  a = a->prox;
        a->prox = p->prox;
    }
} free(p);
```

Resultado

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(4.000000 + i5.000000)
```

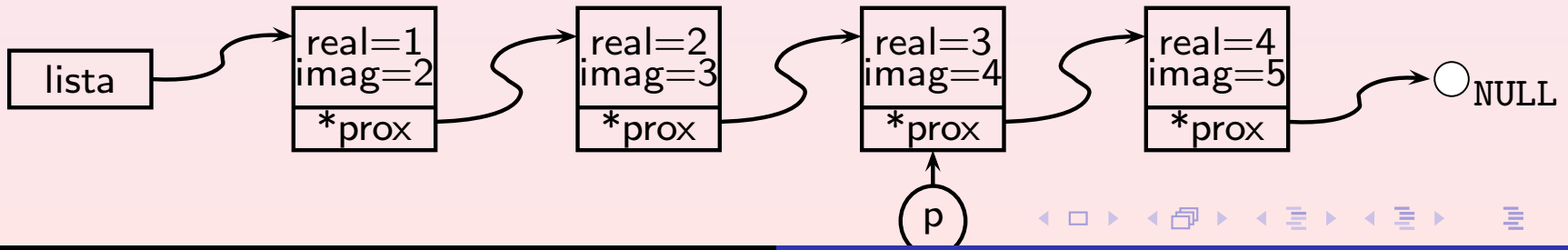


Removendo um elemento do meio pro fim

```
if(p != NULL) {
    if(p == lista) {
        lista = p->prox;
    } else {
        a = lista;
        while(a->prox != p)  a = a->prox;
        a->prox = p->prox;
    }
} free(p);
```

Resultado

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(4.000000 + i5.000000)
```

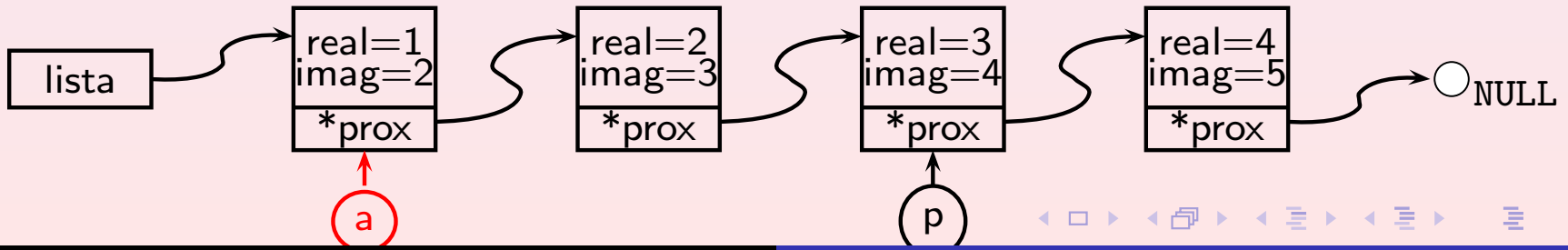


Removendo um elemento do meio pro fim

```
if(p != NULL) {
    if(p == lista) {
        lista = p->prox;
    } else {
        a = lista;
        while(a->prox != p)  a = a->prox;
        a->prox = p->prox;
    }
} free(p);
```

Resultado

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(4.000000 + i5.000000)
```

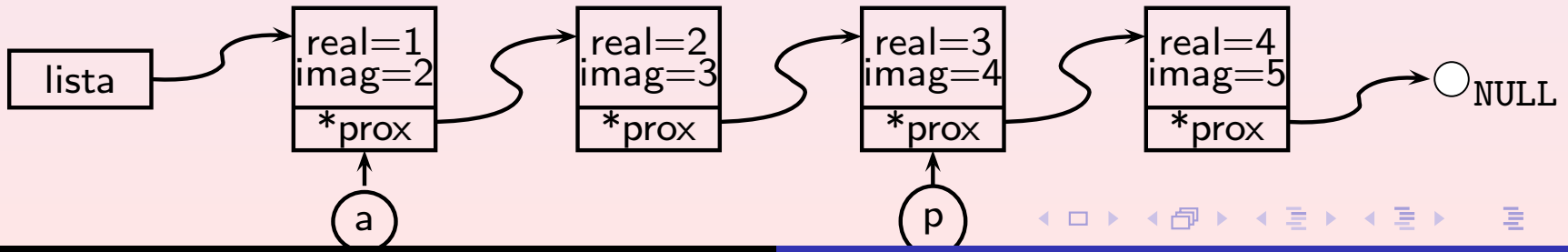


Removendo um elemento do meio pro fim

```
if(p != NULL) {
    if(p == lista) {
        lista = p->prox;
    } else {
        a = lista;
        while(a->prox != p)  a = a->prox;
        a->prox = p->prox;
    }
} free(p);
```

Resultado

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(4.000000 + i5.000000)
```

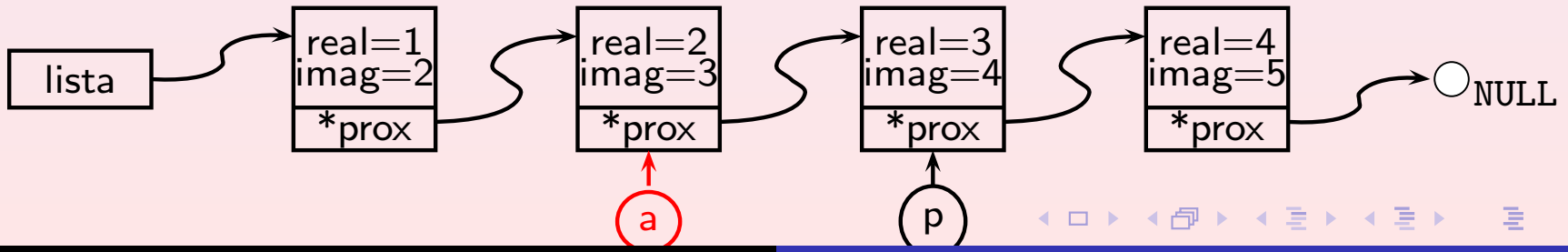


Removendo um elemento do meio pro fim

```
if(p != NULL) {
    if(p == lista) {
        lista = p->prox;
    } else {
        a = lista;
        while(a->prox != p)  a = a->prox;
        a->prox = p->prox;
    }
} free(p);
```

Resultado

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(4.000000 + i5.000000)
```

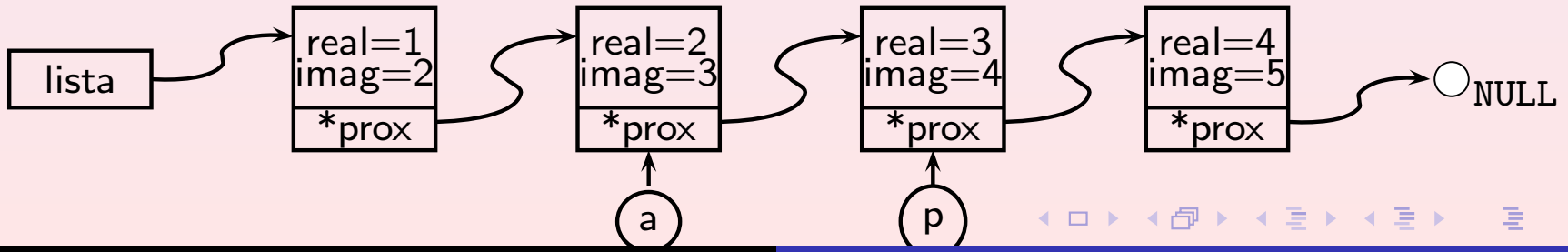


Removendo um elemento do meio pro fim

```
if(p != NULL) {
    if(p == lista) {
        lista = p->prox;
    } else {
        a = lista;
        while(a->prox != p)  a = a->prox;
        a->prox = p->prox;
    }
} free(p);
```

Resultado

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(4.000000 + i5.000000)
```

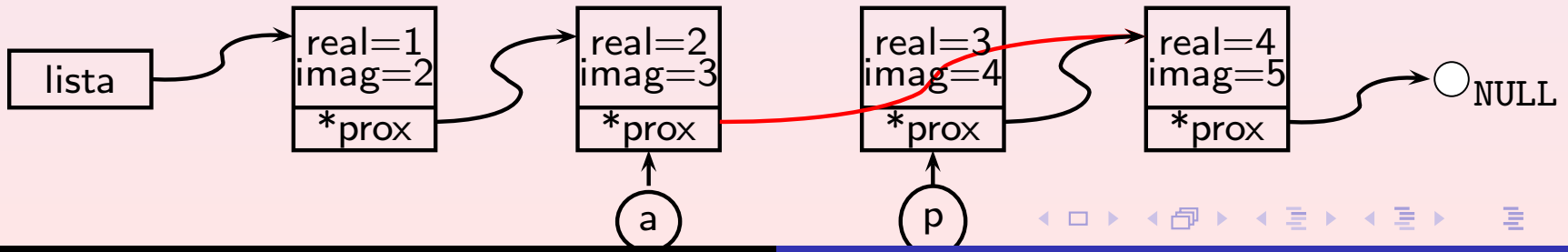


Removendo um elemento do meio pro fim

```
if(p != NULL) {
    if(p == lista) {
        lista = p->prox;
    } else {
        a = lista;
        while(a->prox != p)  a = a->prox;
        a->prox = p->prox;
    }
} free(p);
```

Resultado

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(4.000000 + i5.000000)
```

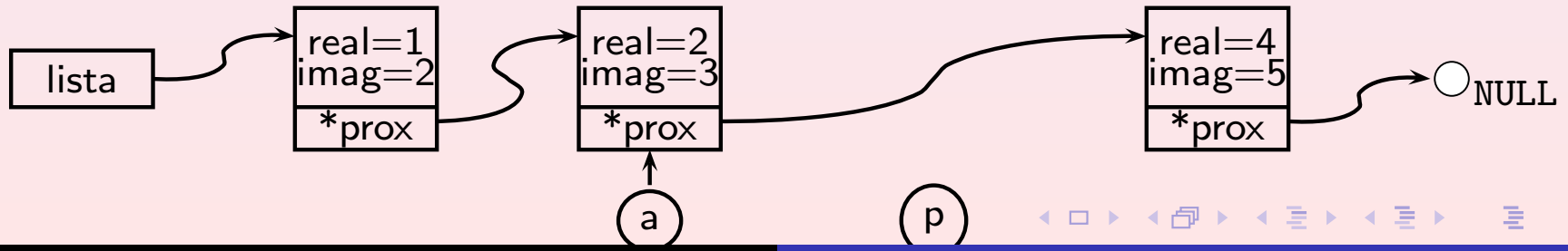


Removendo um elemento do meio pro fim

```
if(p != NULL) {
    if(p == lista) {
        lista = p->prox;
    } else {
        a = lista;
        while(a->prox != p)  a = a->prox;
        a->prox = p->prox;
    }
} free(p);
```

Resultado

```
(1.000000 + i2.000000)
(2.000000 + i3.000000)
(4.000000 + i5.000000)
```



Trocando o primeiro elemento, a, com outro qualquer, b

```
if(b != NULL) {
    c = a->prox;
    d = lista;
    while(d->prox != b) d=d->prox;
    if(a == lista) {
        lista = b;
        a->prox = b->prox;
        if(b == c)
            b->prox = a;
        else
            b->prox = c;
        if(a != d)
            d->prox = a;
    }
}
```

Resultado para b=lista->prox->prox

```
(3.000000 + i4.000000)
(2.000000 + i3.000000)
(1.000000 + i2.000000)
(4.000000 + i5.000000)
```

Trocando dois elementos quais quer, a e b, a antecede b

```
if(b != NULL) {
    d = a->prox;
    e = lista;
    while(e->prox != b) e=e->prox;
    if(a == lista)
        lista = b;
    else {
        c = lista;
        while(c->prox != a) c = c->prox;
        c->prox = b;
    }
    a->prox = b->prox;
    if(b == d)
        b->prox = a;
    else
        b->prox = d;
    if(a != e)
        e->prox = a;
}
```

Resultado

```
(1.000000 + i2.000000)
(4.000000 + i5.000000)
(3.000000 + i4.000000)
(2.000000 + i3.000000)
```

Variações sobre a lista encadeada

- Lista Circular
  - O último elemento, ao invés de ser aterrado ao NULL pode apontar para o primeiro elemento.
  - Quando a lista não é circular, precisamos de dois ponteiros, a "cabeça", que aponta para o primeiro elemento e um ponteiro de navegação que sempre vai do primeiro ao último.
  - Na lista circular, basta o ponteiro de navegação, pois não há cabeça na lista.
  - Para inserir um novo elemento na lista, o algoritmo é mais simples se for inserido na posição seguinte à que o ponteiro aponta.
  - Remoção e troca sempre é necessário que os ponteiros auxiliares dêem a volta na lista para saber que aponta para a posição corrente.

Atividades

- Simule (ou teste) os algoritmos em casos limites, quando aplicável: Lista vazia, só um elemento, só dois. Ponteiro está na cabeça, ponteiro aponta para o último, ponteiro aponta para NULL.
- Construa os algoritmos de Inserção, Remoção e Troca para o caso de lista circular.