Aula 11-A

PONTEIROS

Conteúdo

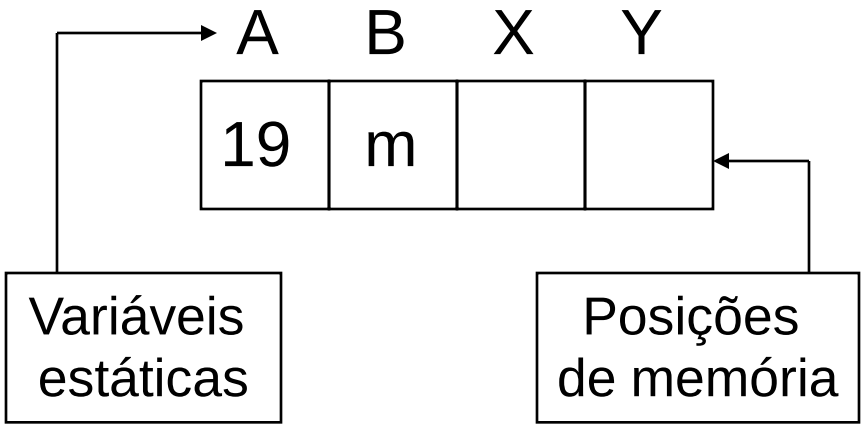
Ponteiros (Pointers)

Ponteiro (Pointer)

Em Pascal, as variáveis e estruturas de dados podem ser estáticas ou dinâmicas.

Variáveis e estruturas estáticas são declaradas na área **var,** possuem um nome próprio, possuem tamanho fixo e existem enquanto o procedimento ou programa no qual foram declaradas estiver sendo executado.

```
var   A : integer;
      B : char;
      X : ...
      Y : ...
```



Ponteiro (Pointer)

Variáveis e estruturas dinâmicas não são declaradas na área **var** e não possuem um nome próprio.

Estruturas de dados dinâmicas, como listas encadeadas, podem crescer ou diminuir. Referências às variáveis dinâmicas são feitas através de ponteiros.

Um ponteiro é uma variável cujo conteúdo é o endereço de uma posição de memória.

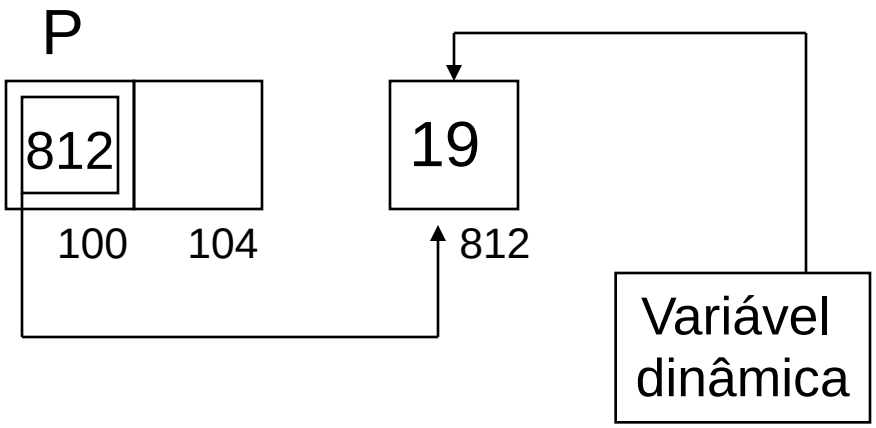
Ponteiro (Pointer)

```
var    P : ^ integer;
```

P é um ponteiro para inteiros.

P^ representa o conteúdo da posição de memória (variável dinâmica) apontada por P.

Neste exemplo, P^ vale 19.

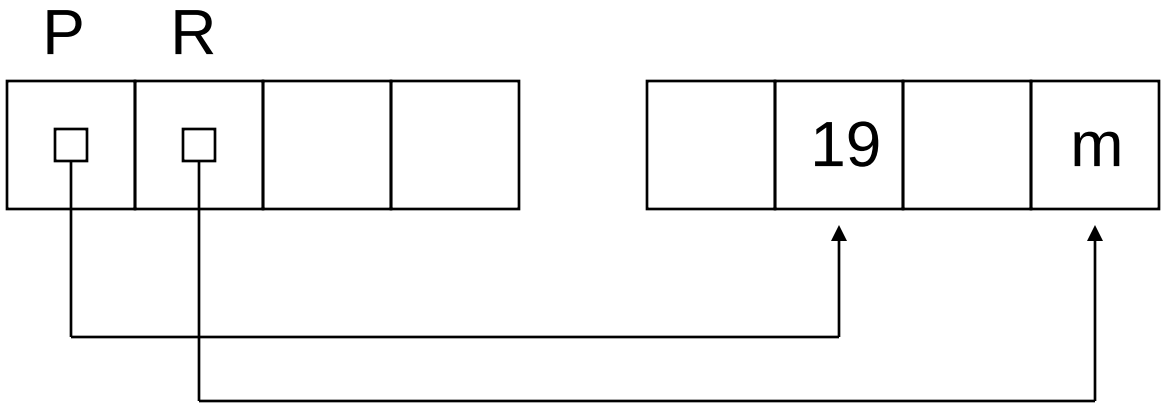


Ponteiro (Pointer)

Um ponteiro é declarado informando-se o tipo da variável por ele apontada.

```
var   P : ^ integer;
      R : ^ char;
```

P é um ponteiro para uma variável dinâmica do tipo inteiro.

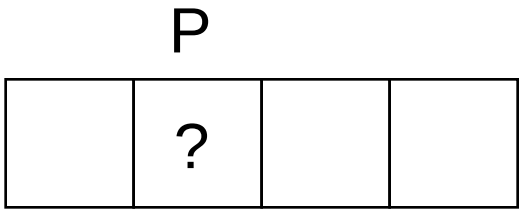


P^ representa o conteúdo da posição apontada por P. P^ vale 19.

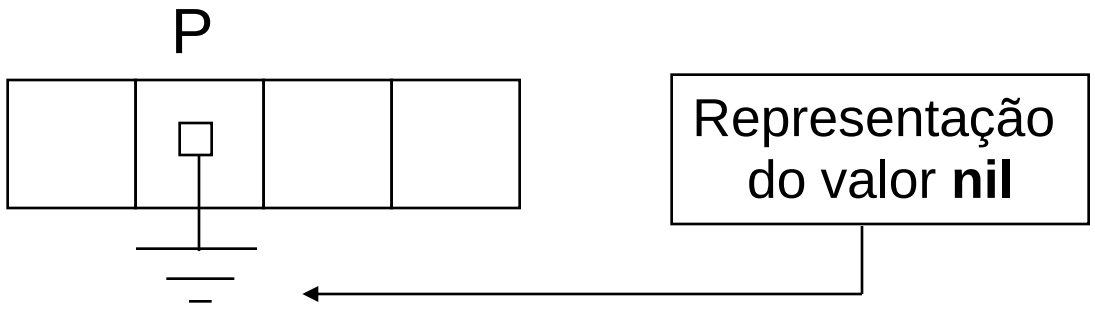
R^ representa o conteúdo da posição apontada por R. R^ vale ‘m’.

Ponteiro (Pointer)

Antes de ser inicializado, o valor de um ponteiro é indefinido. Este valor será representado por ‘?’.

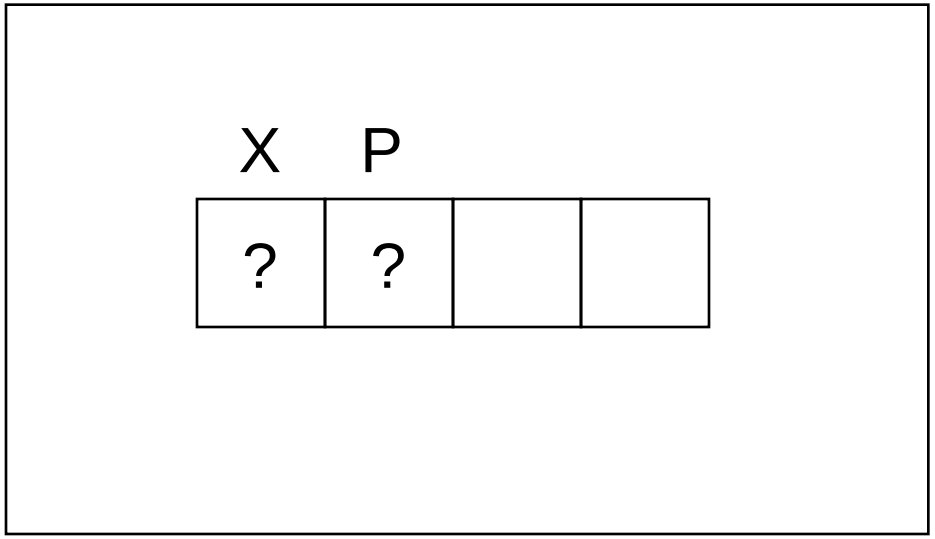


Existe uma constante predefinida do tipo ponteiro, chamada **nil**, que indica que o ponteiro não está referenciando nenhuma posição de memória.



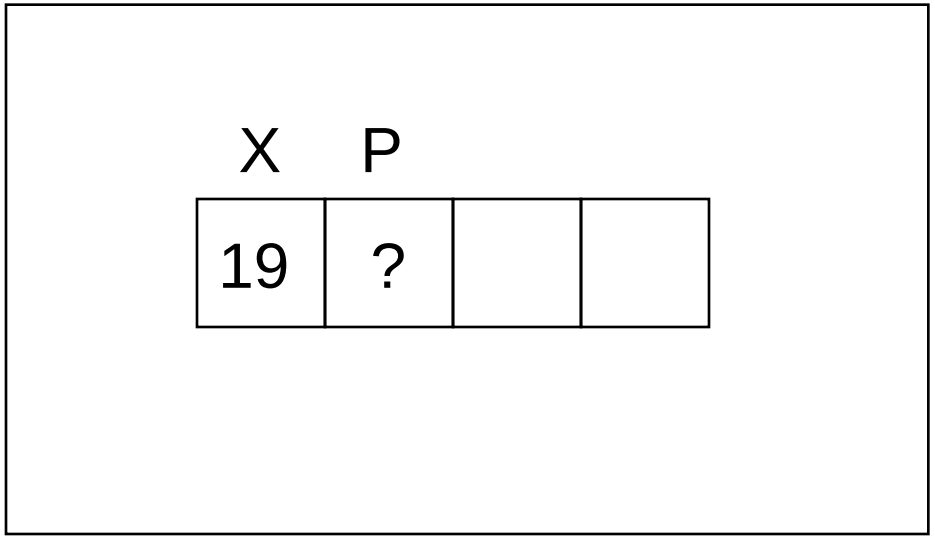
Ponteiro (Pointer)

```
var
      X :  integer;
      P : ^ integer;

begin
       X := 19;
       P := nil;
. . .
end.
```



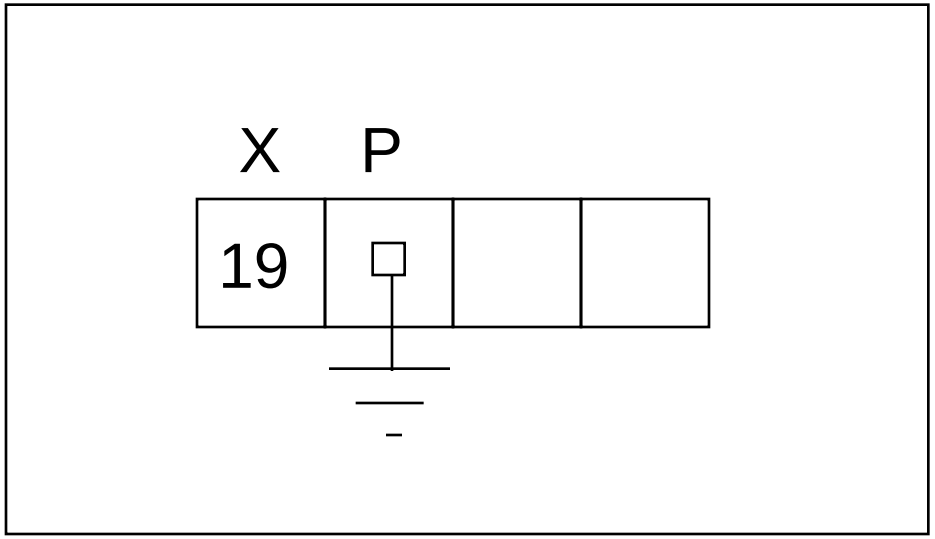
Ponteiro (Pointer)

```
var
      X :  integer;
      P : ^ integer;

begin
       X := 19;
       P := nil;
. . .
end.
```



Ponteiro (Pointer)

```
var
      X :  integer;
      P : ^ integer;

begin
       X := 19;
       P := nil;
. . .
end.
```

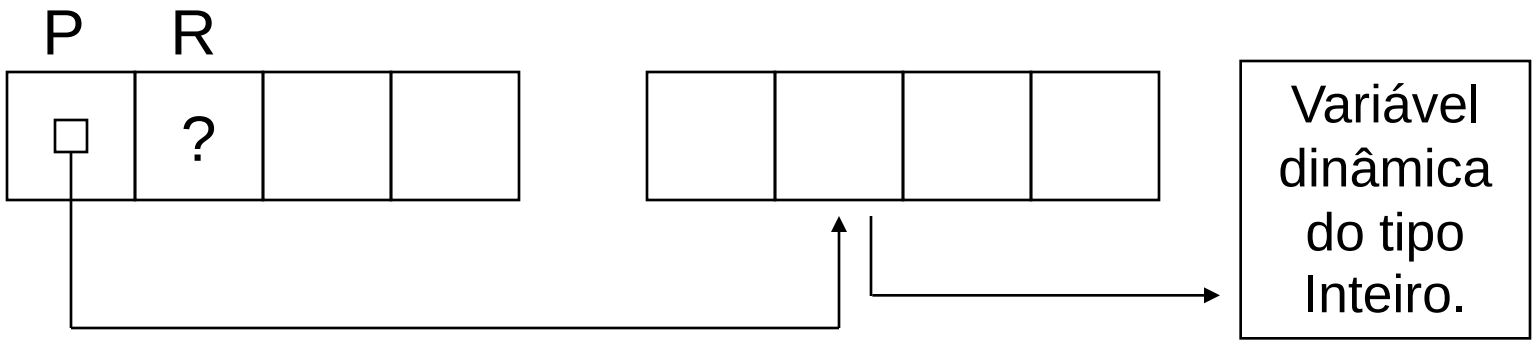


Operador new

O operador **new**, que recebe uma variável ponteiro **P** como parâmetro, cria uma variável dinâmica apontada pelo ponteiro **P**. O ponteiro **P** passa a ter o endereço da variável criada.

```
var    P : ^ integer;
. . .
new (P);
```

O tipo da variável dinâmica será o mesmo o tipo especificado na declaração de P.



Operador new

Alocação dinâmica de memória é o processo de reservar espaço de memória ao longo da execução do programa, para a criação de variáveis e estruturas de dados dinâmicas.

Alocação estática de memória é feita para as variáveis e estruturas de dados declaradas previamente no programa e nos subprogramas.

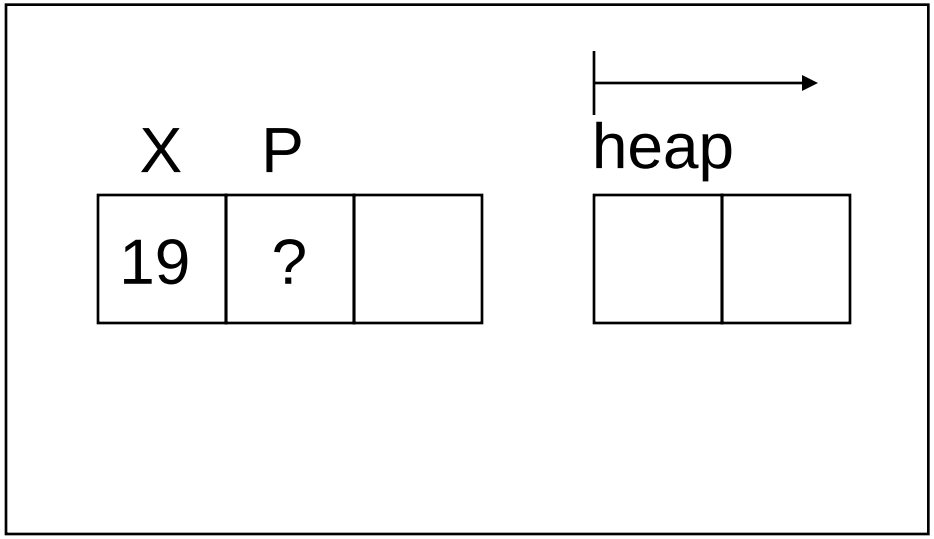
Operador new

A Alocação dinâmica se dá em uma área de memória especial, gerenciada pelo Pascal, chamada **heap**.

Toda vez que a operação new(**P**) é executada, uma área de memória do heap, correspondente ao tipo da variável ou estrutura apontada por **P**, é alocada ao programa, deixando de fazer parte do **heap**.

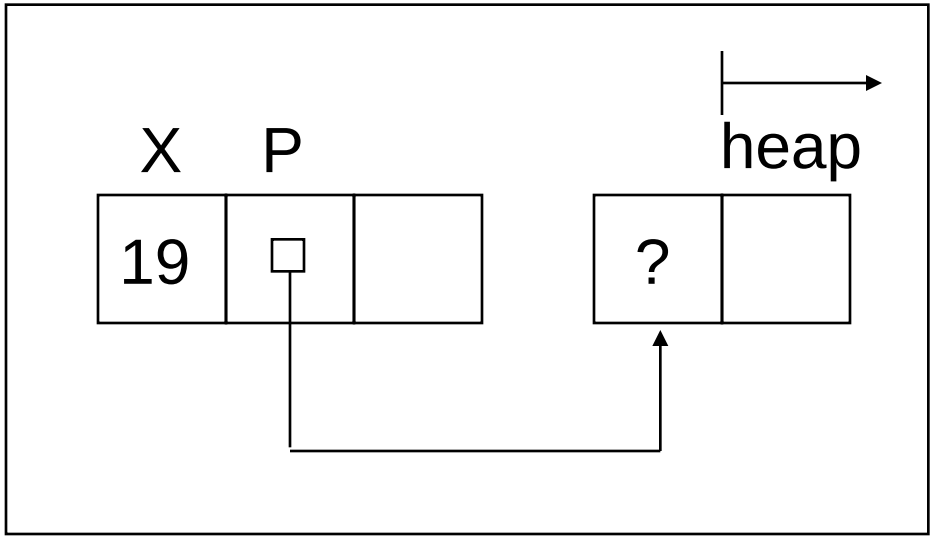
Operador new

```
var
      X :  integer;
      P : ^ integer;

begin
       X := 19;
       new (P);
. . .
end.
```



Operador new

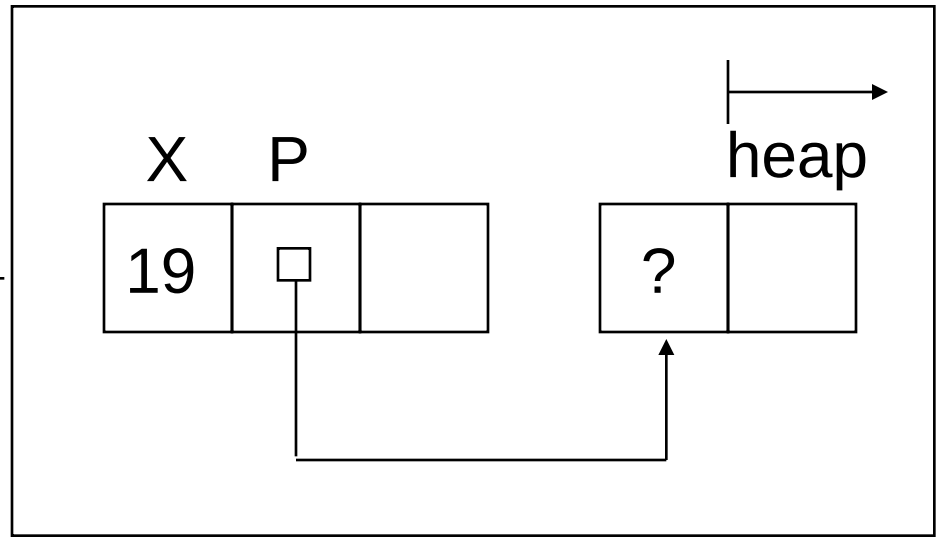
```
var
      X :  integer;
      P : ^ integer;

begin
       X := 19;
       new (P);
. . .
end.
```



Operador new

A variável criada com o operador **new** (P) passa a ser referenciada por P^, que representa a variável (o conteúdo de memória) apontada por P

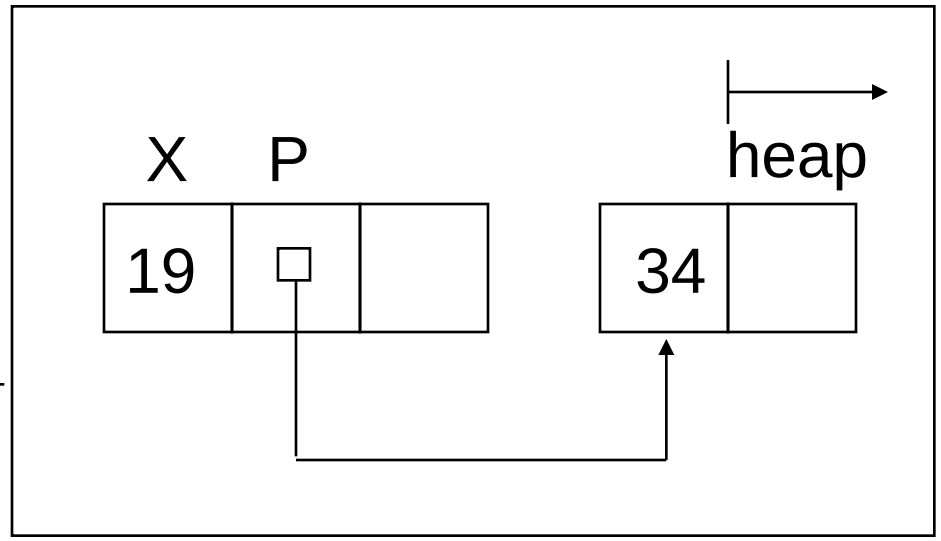
```
begin
    X := 19;
    new (P);
    P^ := 34;
    X := P^ ;
end.
```



Operador new

A variável criada com o operador **new** (P) passa a ser referenciada por P^, que representa a variável (o conteúdo de memória) apontada por P

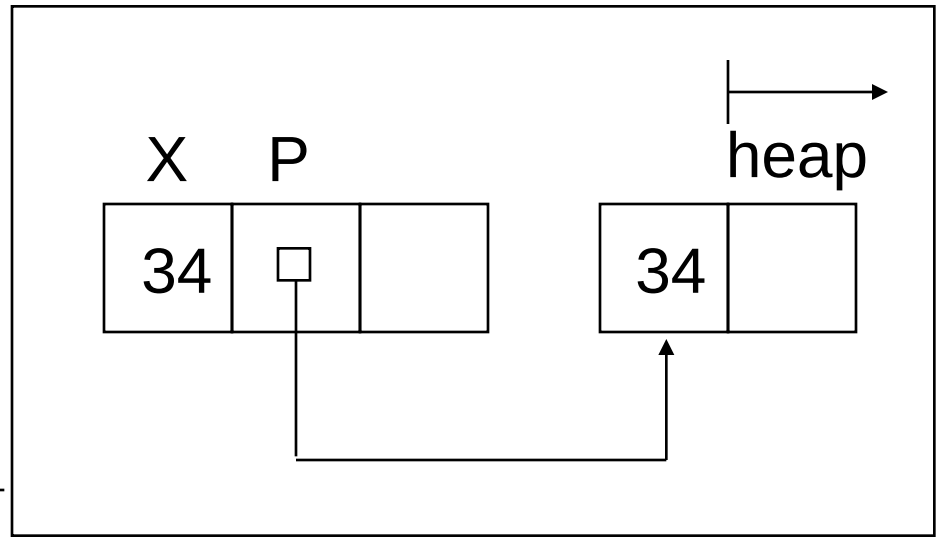
```
begin
    X := 19;
    new (P);
    P^ := 34;
    X := P^ ;
end.
```



Operador new

A variável criada com o operador **new** (P) passa a ser referenciada por P^, que representa a variável (o conteúdo de memória) apontada por P

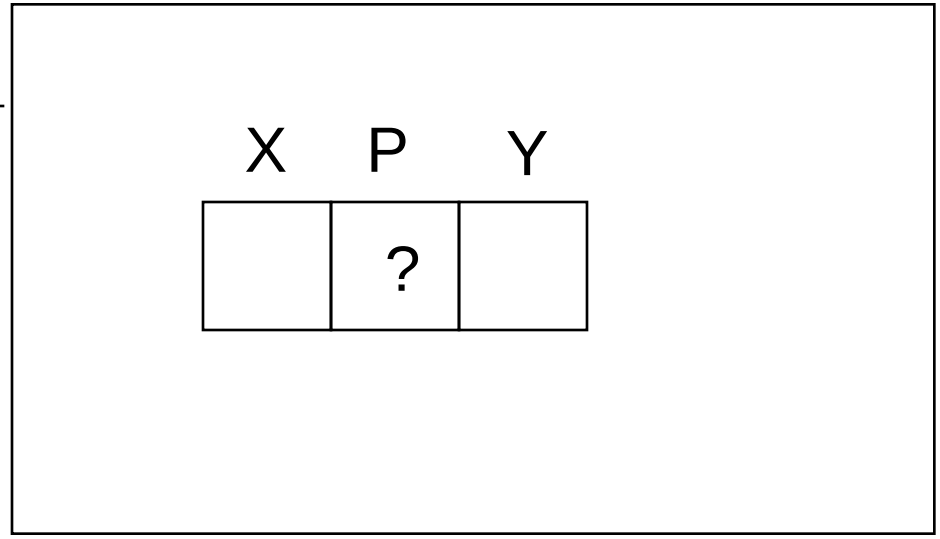
```
begin
    X := 19;
    new (P);
    P^ := 34;
    X := P^ ;
end.
```



Endereço de uma variável

Um ponteiro **P** pode receber o endereço de uma variável **V** através da atribuição **P := @V**. Neste caso, **P** deve ser um ponteiro para uma variável do tipo de **V**.

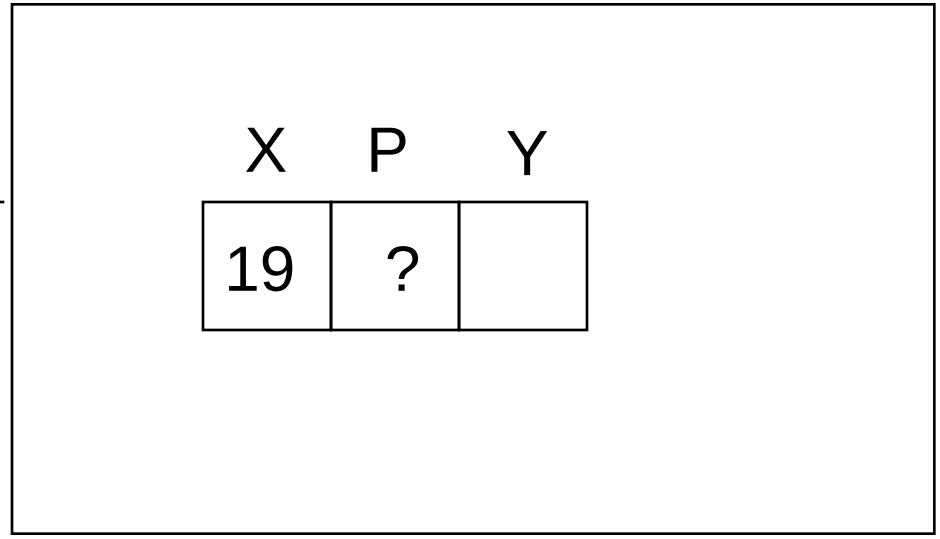
```
begin
    X := 19;
    P := @X;
    Y := P^ ;
end.
```



Endereço de uma variável

Um ponteiro **P** pode receber o endereço de uma variável **V** através da atribuição **P := @V**. Neste caso, **P** deve ser um ponteiro para uma variável do tipo de **V**.

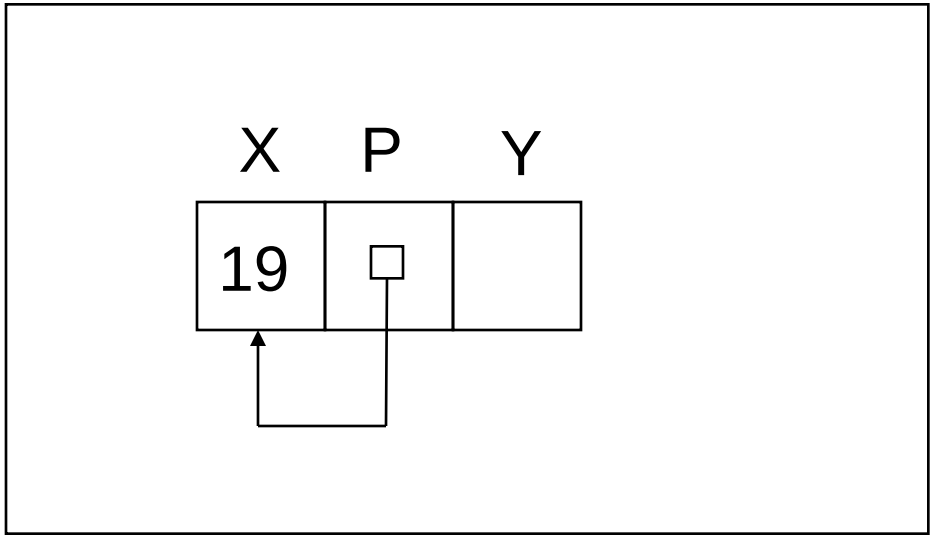
```
begin
    X := 19;
    P := @X;
    Y := P^ ;
end.
```



Endereço de uma variável

Um ponteiro **P** pode receber o endereço de uma variável **V** através da atribuição **P := @V**. Neste caso, **P** deve ser um ponteiro para uma variável do tipo de **V**.

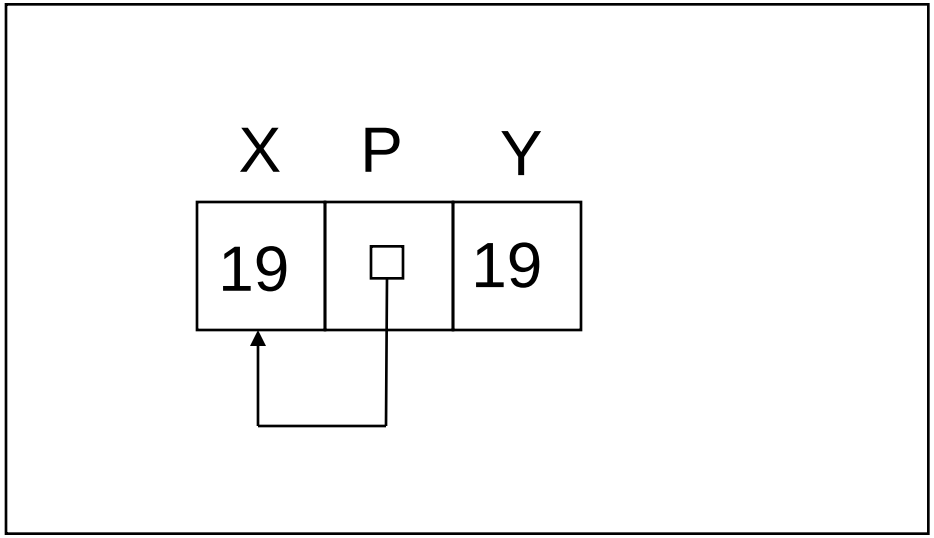
```
begin
    X := 19;
    P := @X;
    Y := P^ ;
end.
```



Endereço de uma variável

Um ponteiro **P** pode receber o endereço de uma variável **V** através da atribuição **P := @V**. Neste caso, **P** deve ser um ponteiro para uma variável do tipo de **V**.

```
begin
    X := 19;
    P := @X;
    Y := P^ ;
end.
```

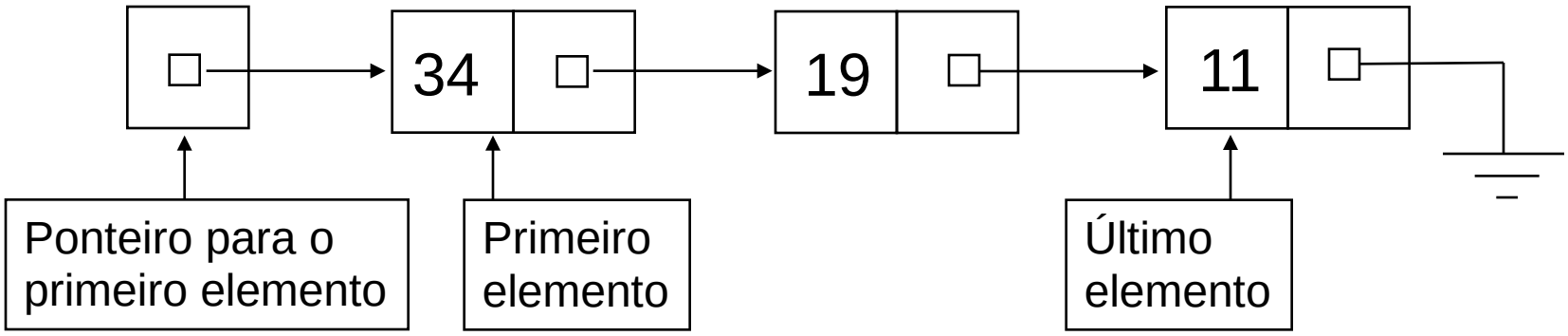


Listas

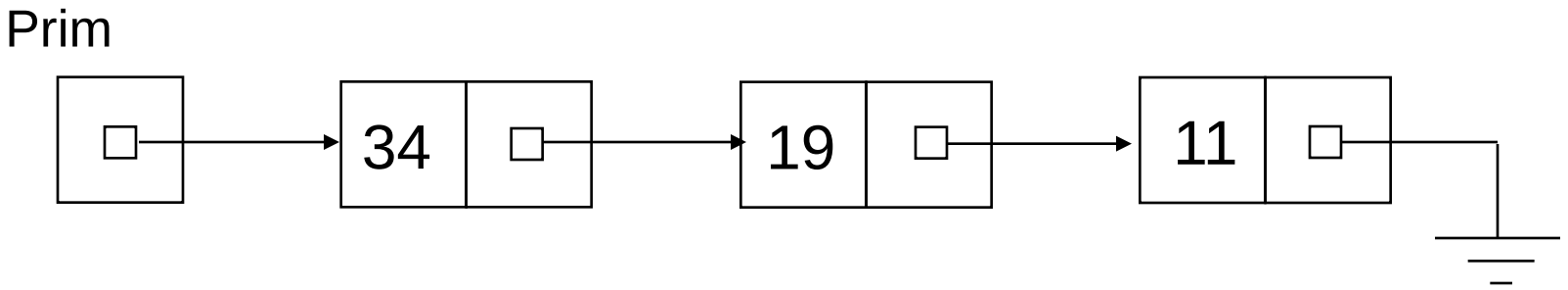
Listas são estruturas de dados comumente implementadas em Pascal utilizando-se ponteiros.

Na sua forma mais simples, uma lista é uma seqüência de elementos encadeados por ponteiros. Uma lista pode ter um número indeterminado de elementos.

O primeiro elemento é referenciado por um ponteiro que indica o início da lista.



Listas



Nesta lista exemplo:

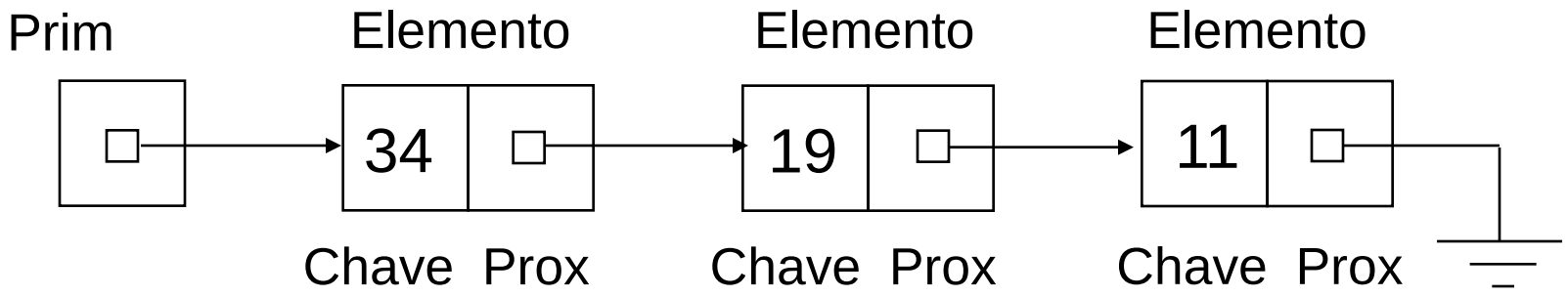
- A variável ponteiro Prim aponta para o primeiro elemento da lista;

- Representa uma seqüência de três elementos: 34, 19 e 11;

- Cada elemento da lista é formado por dois campos: um inteiro e um ponteiro para o próximo elemento;

- O último elemento da lista não aponta para nenhum outro elemento, o seu campo ponteiro contém o valor **nil**.

Listas



Declaração recursiva de tipos e variáveis para implementação da lista acima:

```
type    Informacao = integer;
        Ponteiro = ^ Elemento;
        Elemento = record
                       Chave : Informacao;
                       Prox : Ponteiro
                   end;
var     Prim : Ponteiro;
```

Listas

Existem algumas operações comuns que se realizam sobre listas:

- criação da lista;
- busca por um elemento;
- inserção de um elemento;
- remoção de um elemento;
- processamento de todos os elementos.

Listas

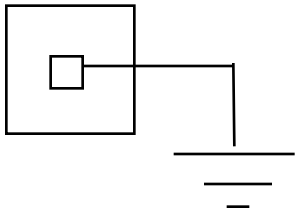
Criação de uma lista, com os elementos 11, 19 e 34.

Listas

Criação de uma lista, com os elementos 11, 19 e 34.

```
Prim := nil;
```

Prim

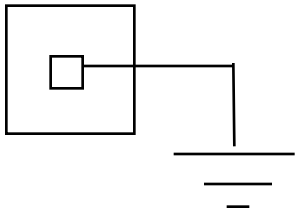


Listas

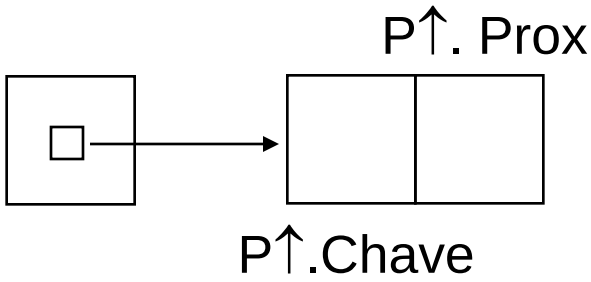
Criação de uma lista, com os elementos 11, 19 e 34.

new (P);

Prim



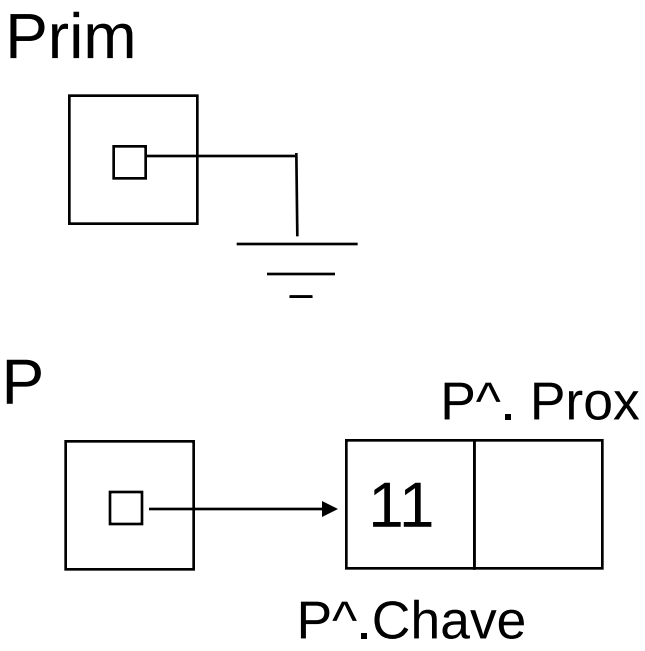
P



Listas

Criação de uma lista, com os elementos 11, 19 e 34.

```
P^.Chave := 11;
```

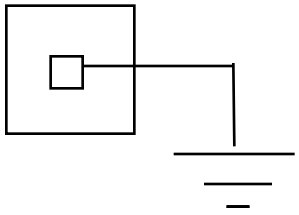


Listas

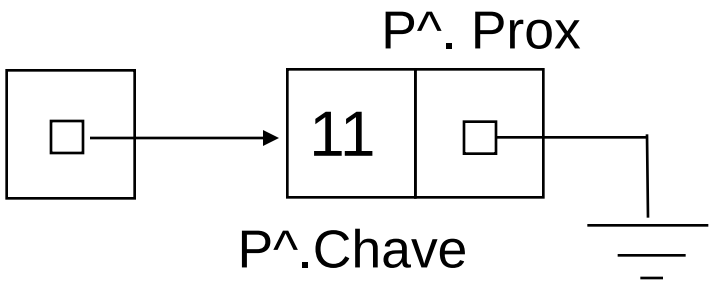
Criação de uma lista, com os elementos 11, 19 e 34.

```
P^.Prox := Prim;
```

Prim



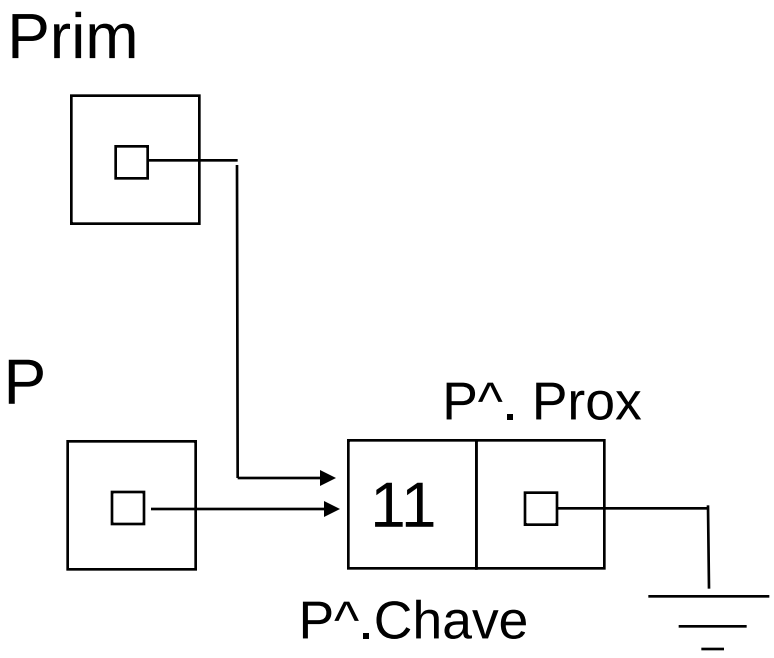
P



Listas

Criação de uma lista, com os elementos 11, 19 e 34.

```
Prim := P;
```

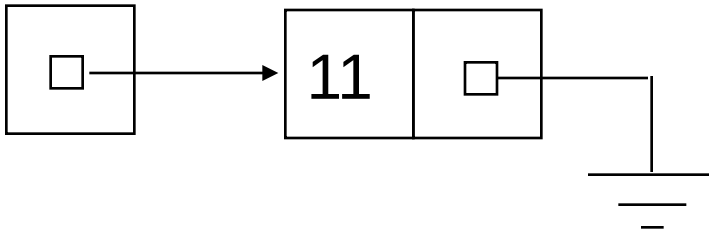


Listas

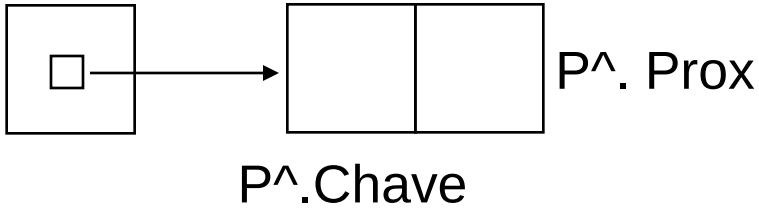
Criação de uma lista, com os elementos 11, 19 e 34.

new (P);

Prim



P

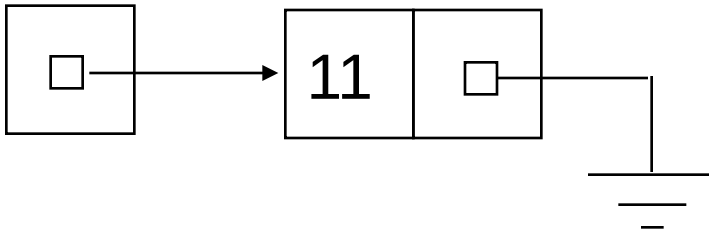


Listas

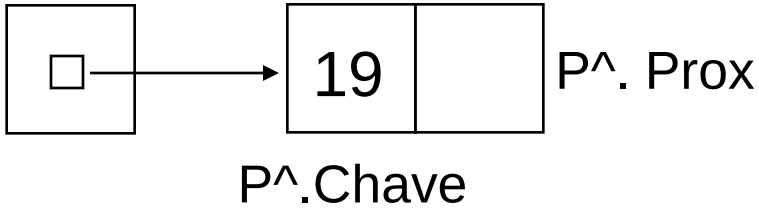
Criação de uma lista, com os elementos 11, 19 e 34.

P^.Chave := 19;

Prim



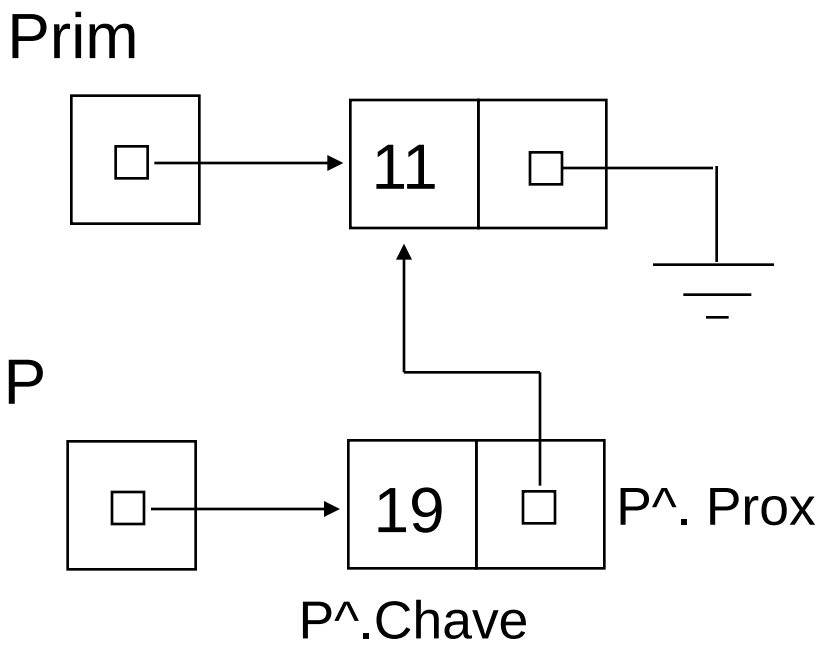
P



Listas

Criação de uma lista, com os elementos 11, 19 e 34.

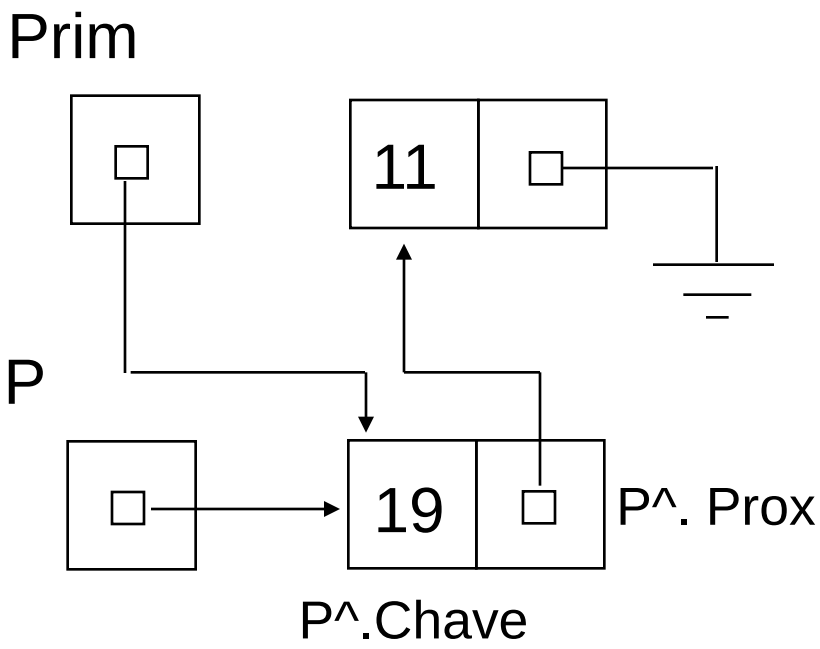
```
P^.Prox := Prim;
```



Listas

Criação de uma lista, com os elementos 11, 19 e 34.

```
Prim := P;
```

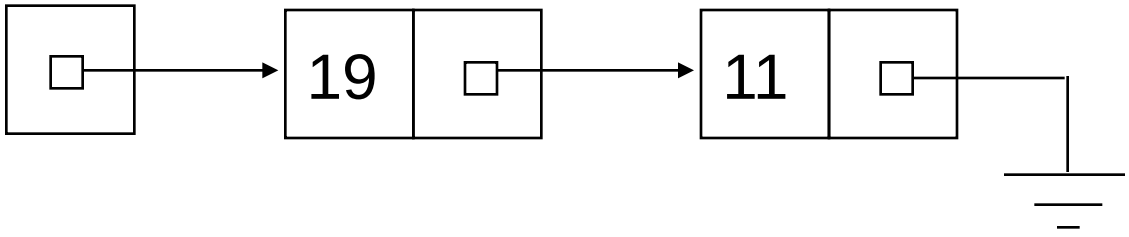


Listas

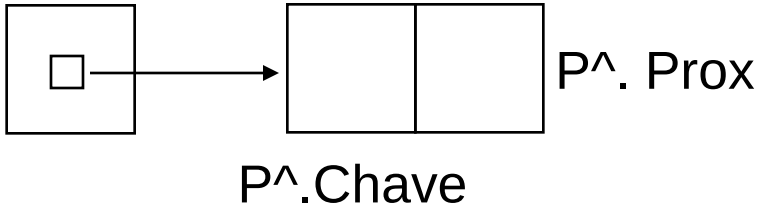
Criação de uma lista, com os elementos 11, 19 e 34.

new (P);

Prim



P

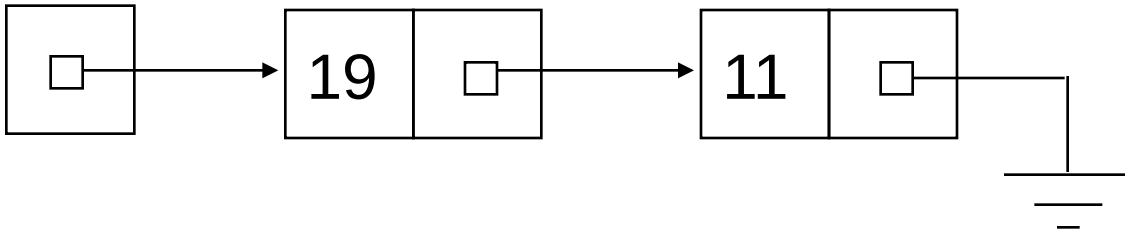


Listas

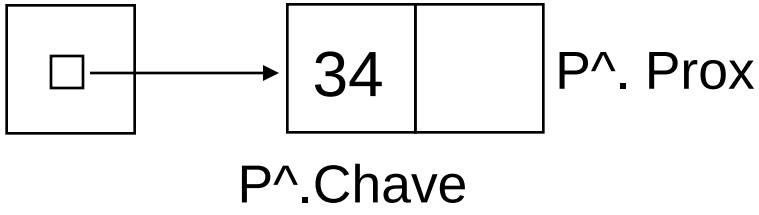
Criação de uma lista, com os elementos 11, 19 e 34.

P^.Chave := 34;

Prim



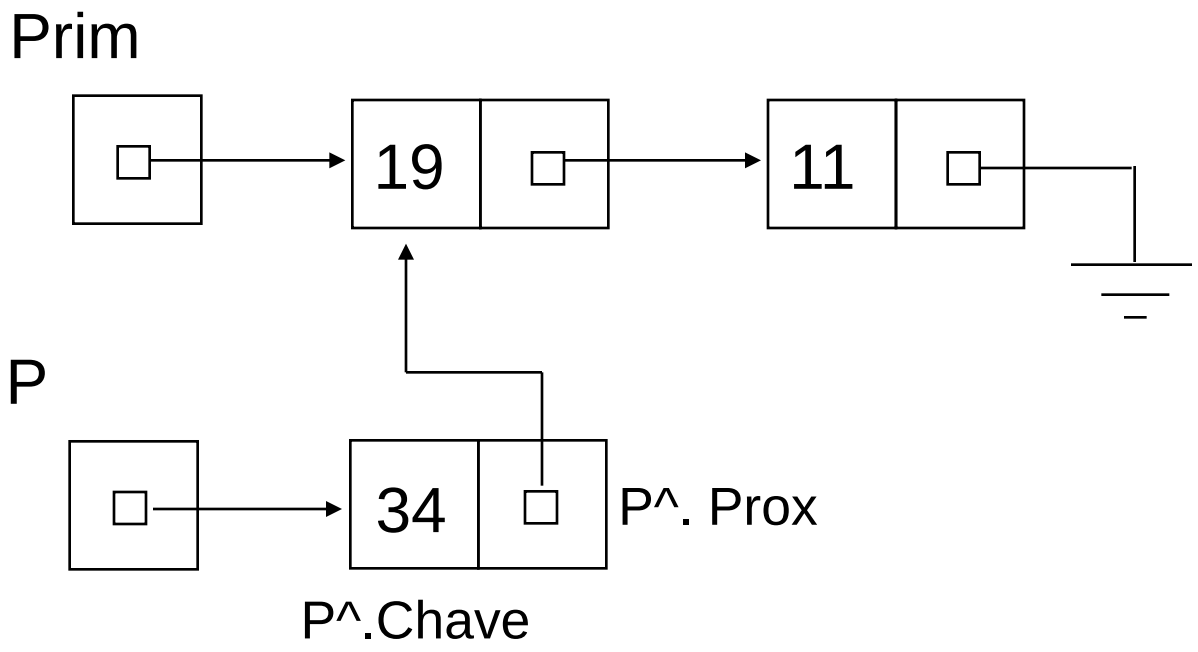
P



Listas

Criação de uma lista, com os elementos 11, 19 e 34.

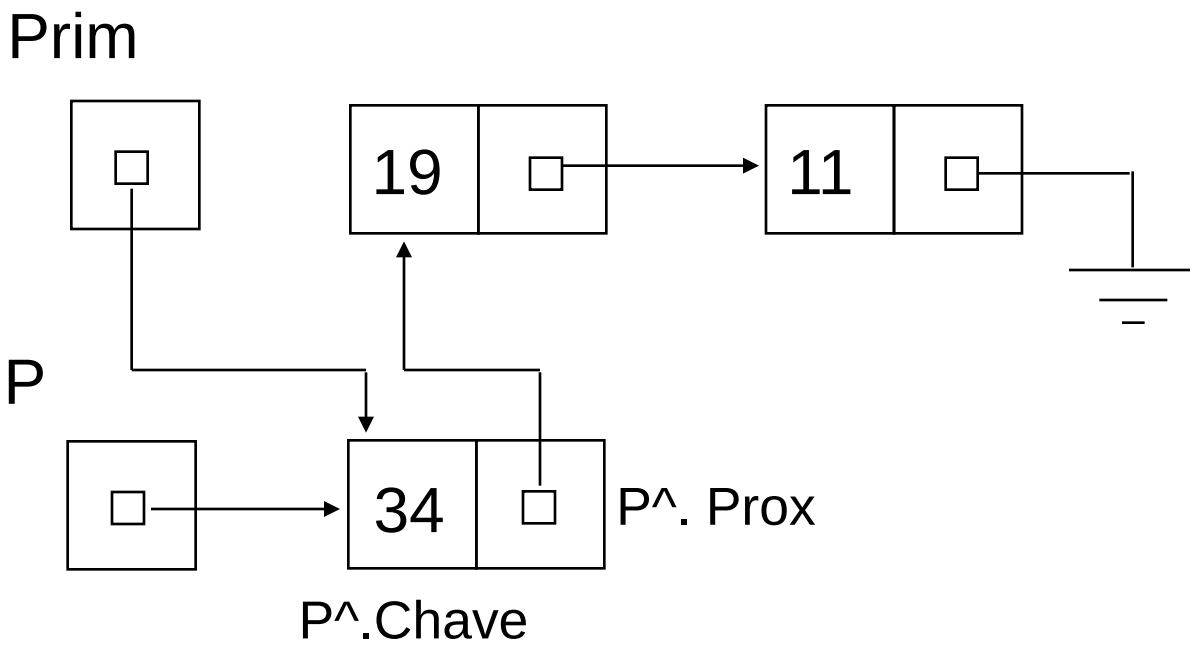
P^.Prox := Prim;



Listas

Criação de uma lista, com os elementos 11, 19 e 34.

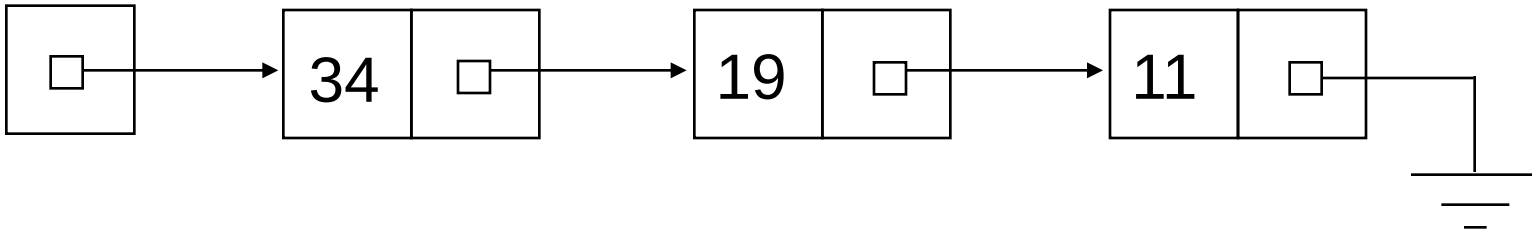
Prim := P;



Listas

Criação de uma lista, com os elementos 11, 19 e 34.

Prim



Listas

Criação de uma lista a partir de valores contidos em um arquivo texto.

```
Procedure CriaLista (Var Arquivo{e} : text; Var Prim{s} : Ponteiro);
var     P : Ponteiro;
Begin
    reset (Arquivo);
    Prim := nil;
    while not eof (Arquivo) do
        begin
            new (P);
            read (Arquivo, P^.Chave);
            P^.Prox := Prim;
            Prim := P
        end;
    close (Arquivo);
end;
```

Listas

Busca por um valor (passado como parâmetro).

```
Procedure ProcuraValor (Prim{e} : Ponteiro; Valor{e}: integer);
var       P : Ponteiro;
          Achou : boolean;
Begin

    P := Prim;

    Achou := false;

    while (P<>nil) and not Achou do
          if (P^.Chave = Valor) then Achou := true
          else   P := P^.Prox

    if Achou then ProcessaElemento(P);

end;
```

Listas

Cuidado com a seguinte simplificação do programa anterior.

```
Procedure ProcuraValor (Prim{e} : Ponteiro; Valor{e}: Informacao);
var       P : Ponteiro;
Begin
    P := Prim;

    while (P<>nil) and (P^.Chave <> Valor) do  P := P^.Prox;

    if (P<>nil) then ProcessaElemento(P);
end;
```

Pois, dependendo do compilador, a condição do **while** pode gerar um erro caso o elemento procurado não se encontre na lista.

Listas

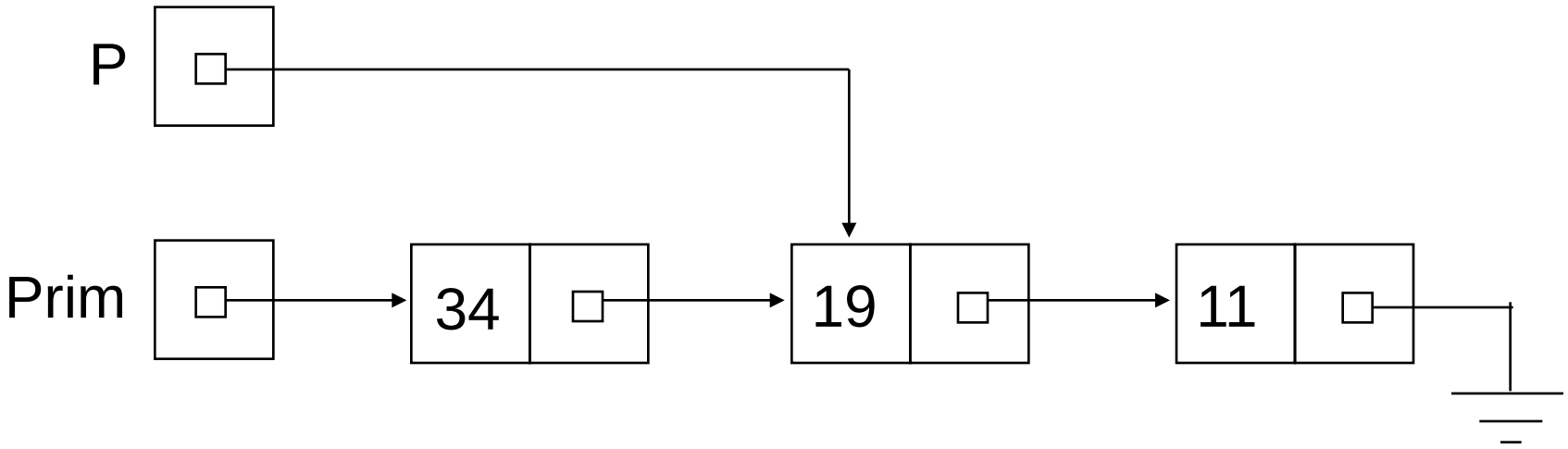
Duas possibilidades devem ser consideradas para executar a **inserção** de um elemento em uma lista:

- inserção **após** o elemento apontado por **P**;

- inserção **antes** do elemento apontado por **P**.

Listas

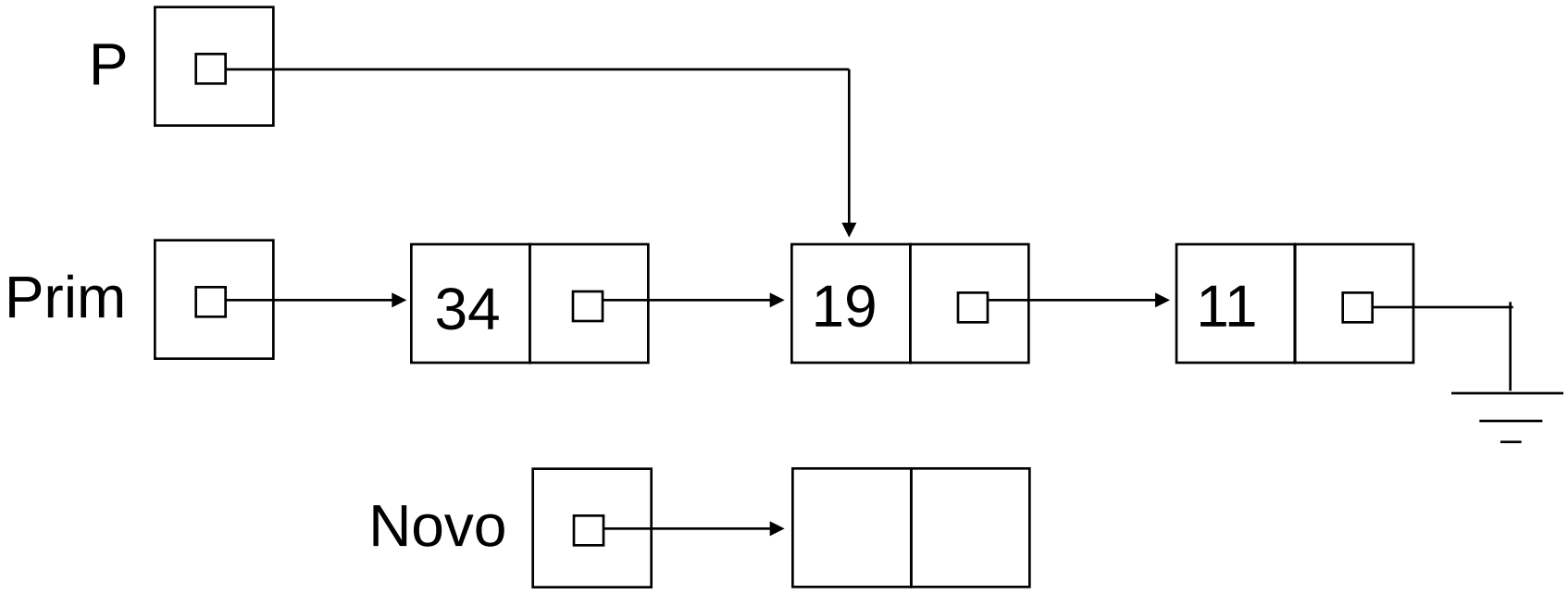
Inserção do elemento 9 após o elemento apontado por P.



Listas

Inserção do elemento 9 após o elemento apontado por P.

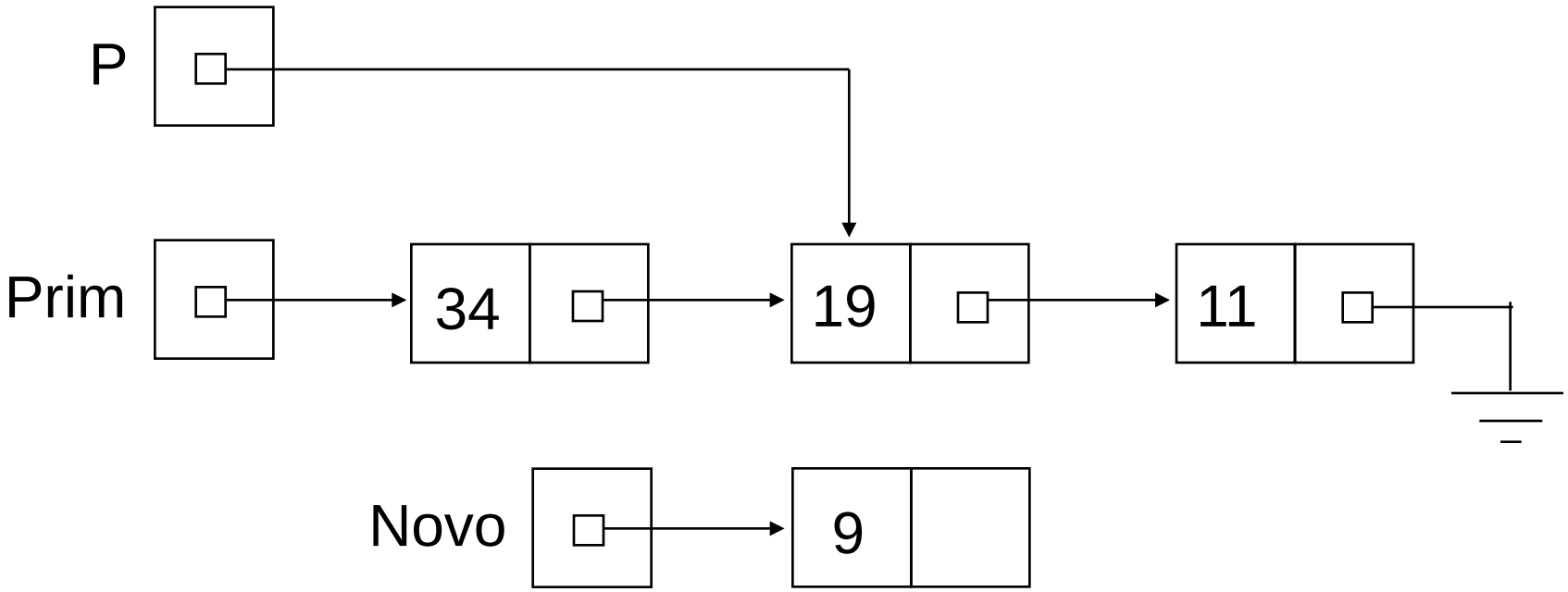
new (Novo);



Listas

Inserção do elemento 9 após o elemento apontado por P.

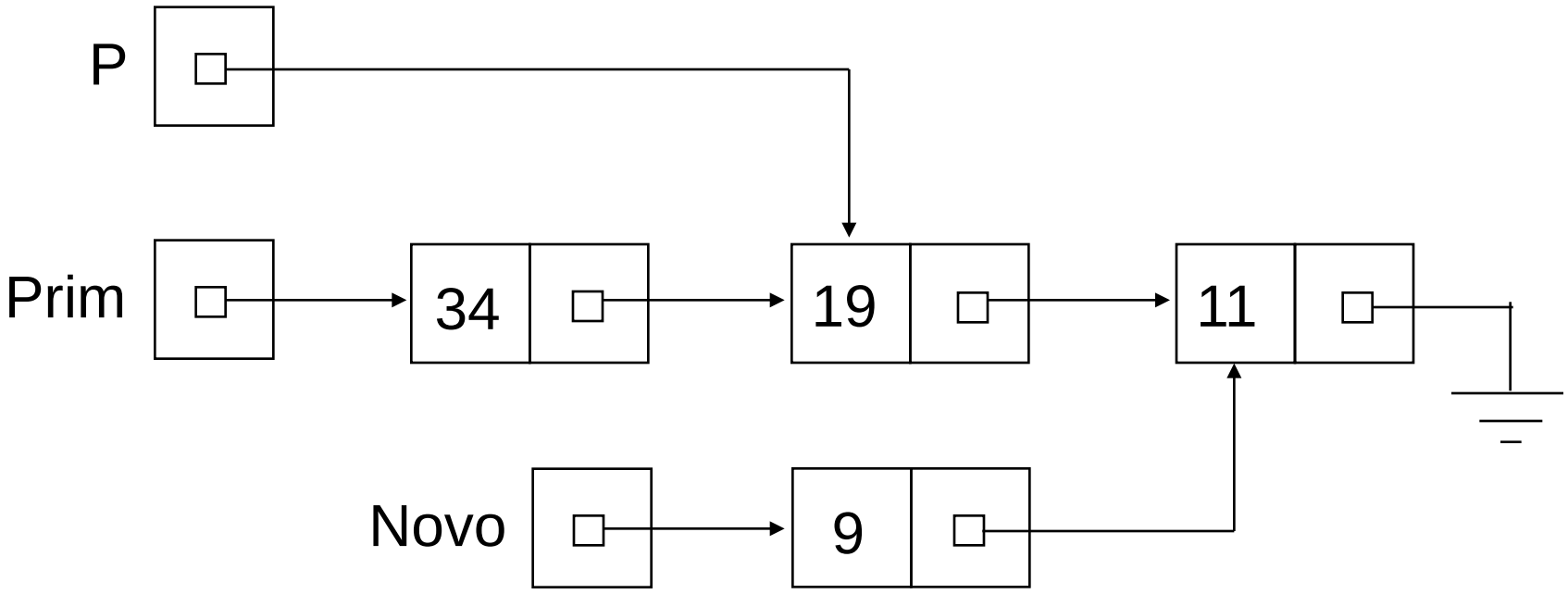
Novo^.Chave := 9;



Listas

Inserção do elemento 9 após o elemento apontado por P.

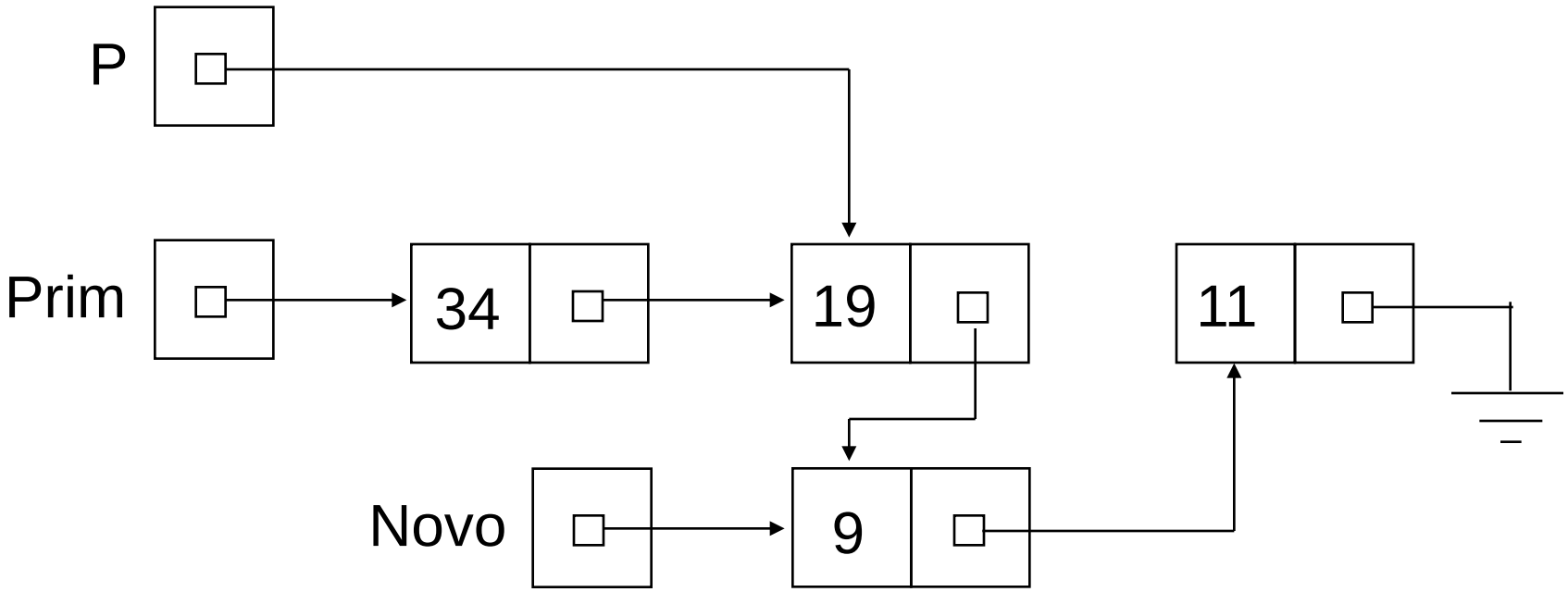
```
Novo^.Prox := P^.Prox;
```



Listas

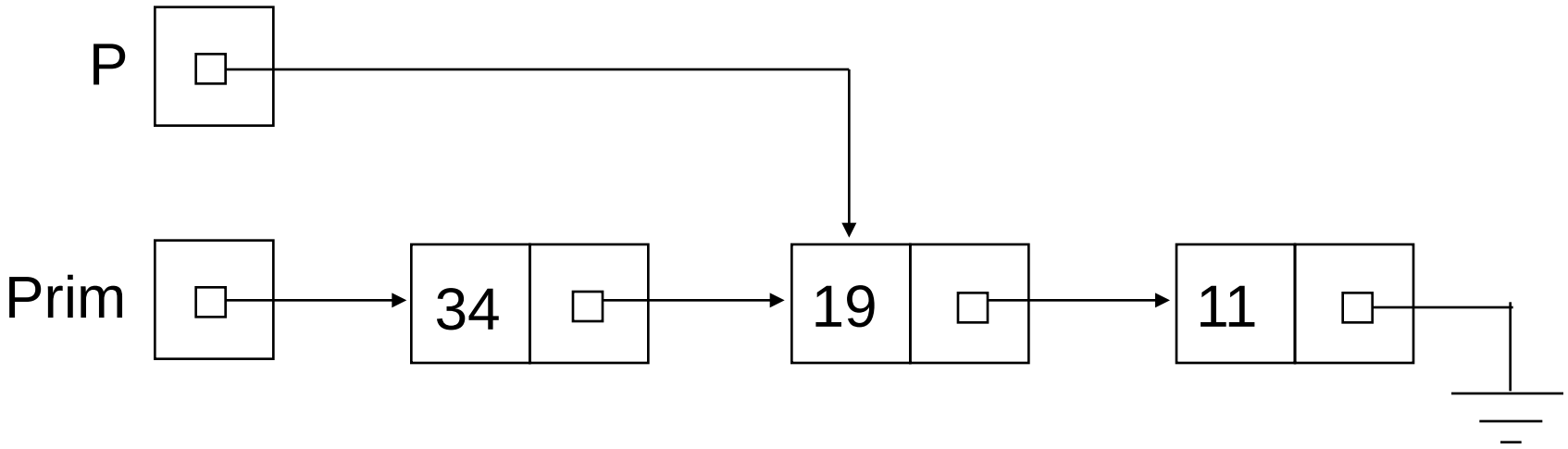
Inserção do elemento 9 após o elemento apontado por P.

P^.Prox := Novo;



Listas

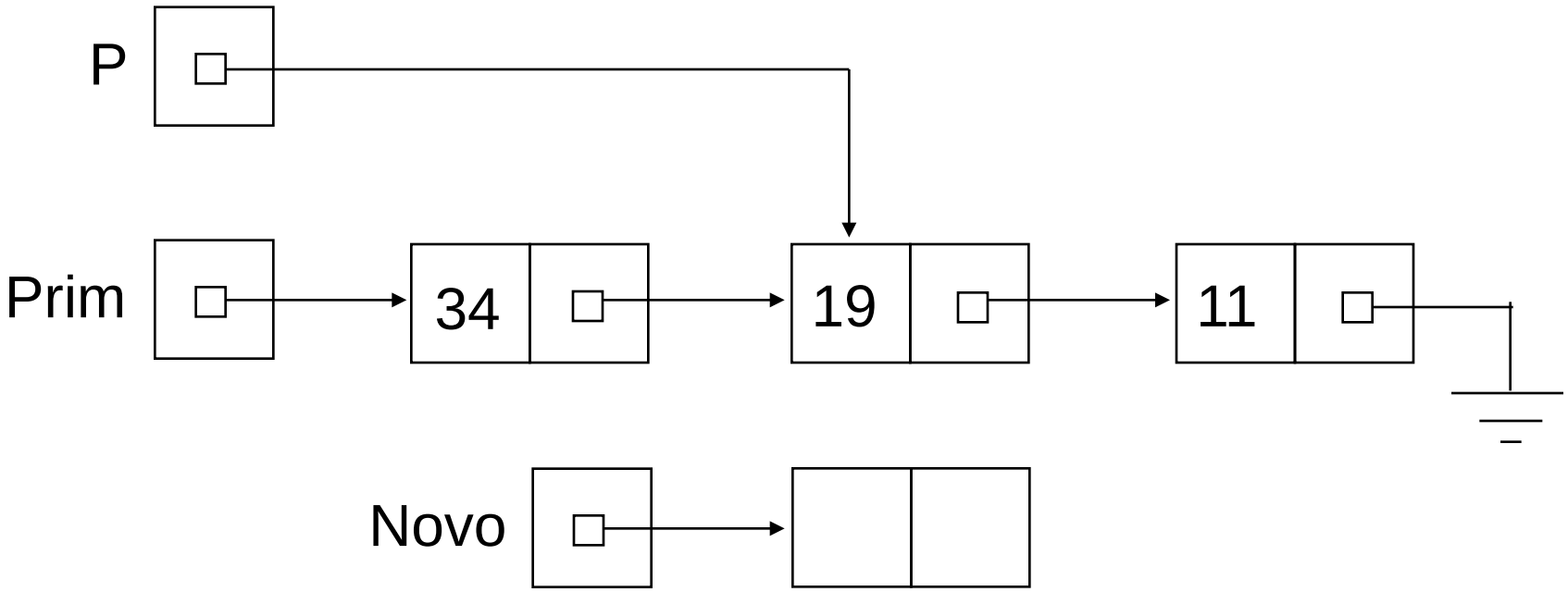
Inserção do elemento 9 antes do elemento apontado por P.



Listas

Inserção do elemento 9 antes do elemento apontado por P.

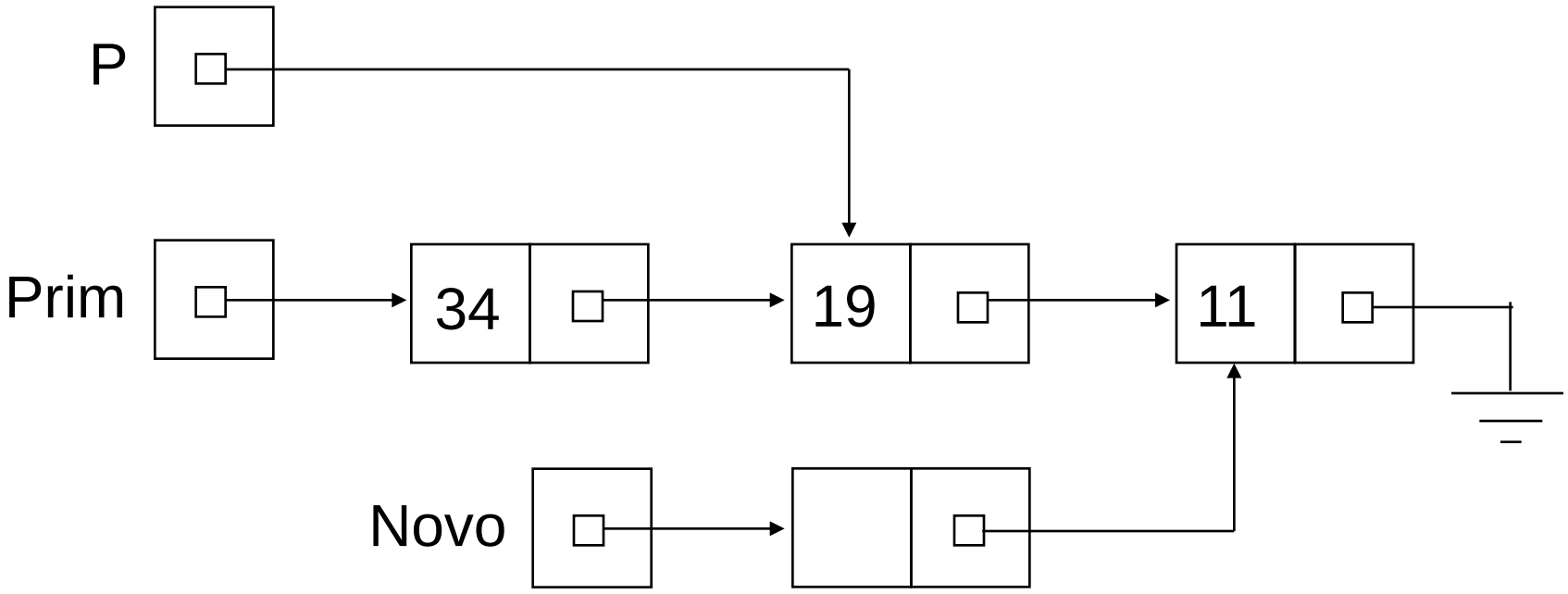
new (Novo);



Listas

Inserção do elemento 9 antes do elemento apontado por P.

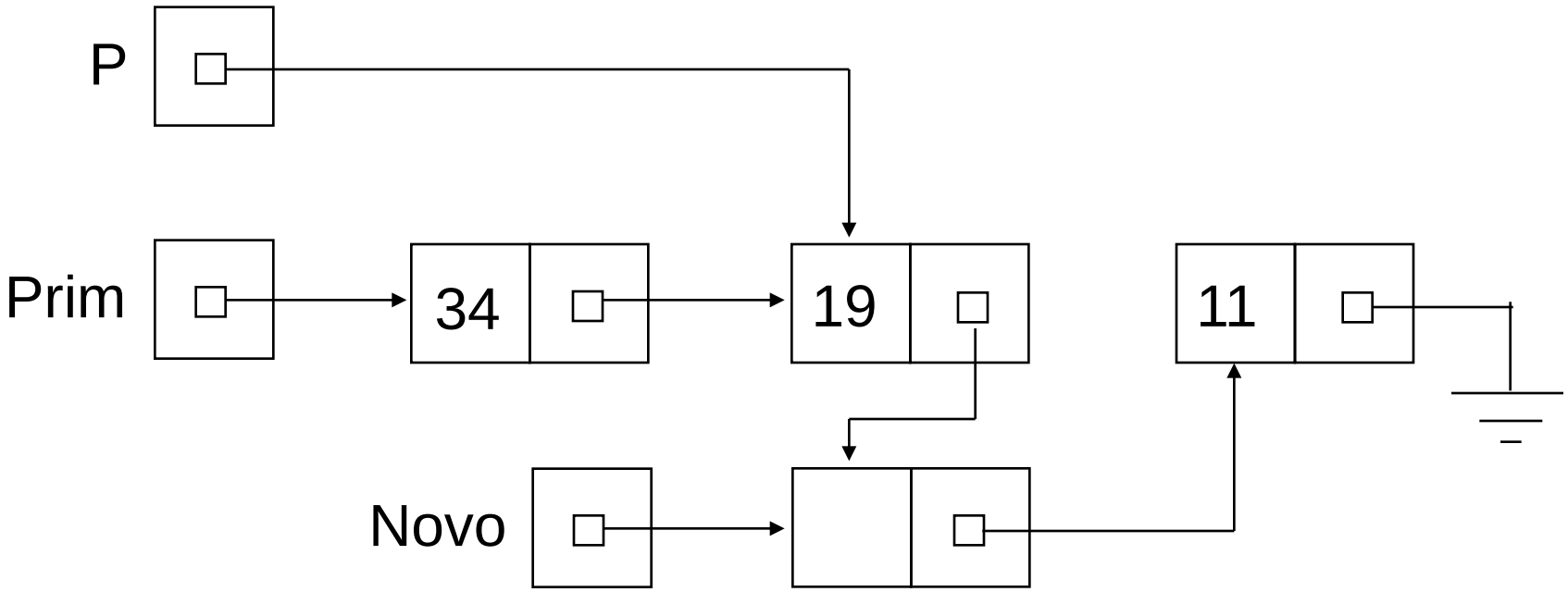
Novo^.Prox := P^.Prox;



Listas

Inserção do elemento 9 antes do elemento apontado por P.

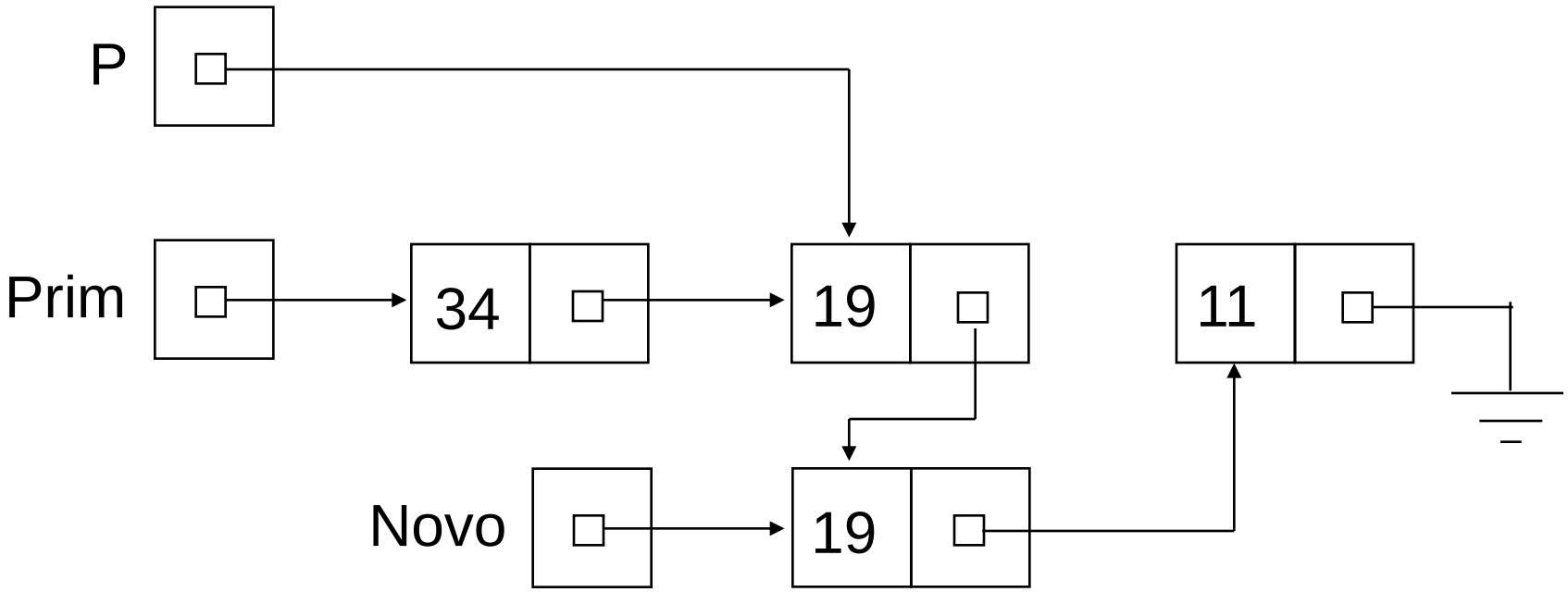
P^.Prox := Novo;



Listas

Inserção do elemento 9 antes do elemento apontado por P.

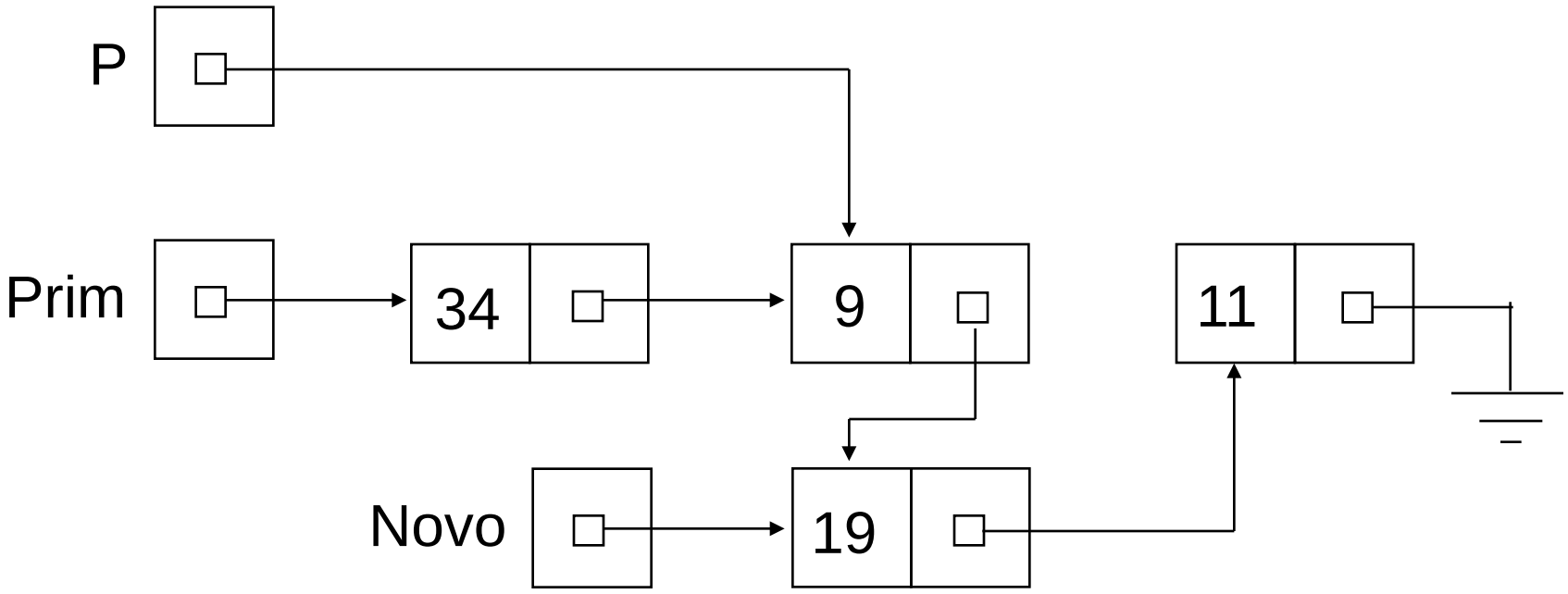
Novo^.Chave := P^.Chave;



Listas

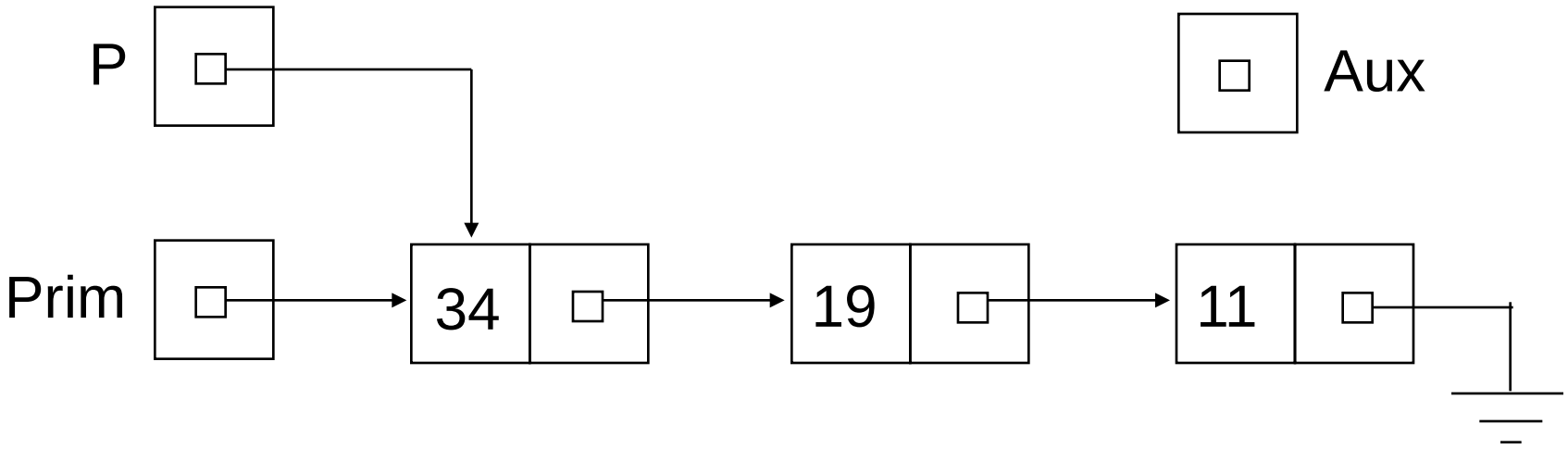
Inserção do elemento 9 antes do elemento apontado por P.

P^.Chave := 9;



Listas

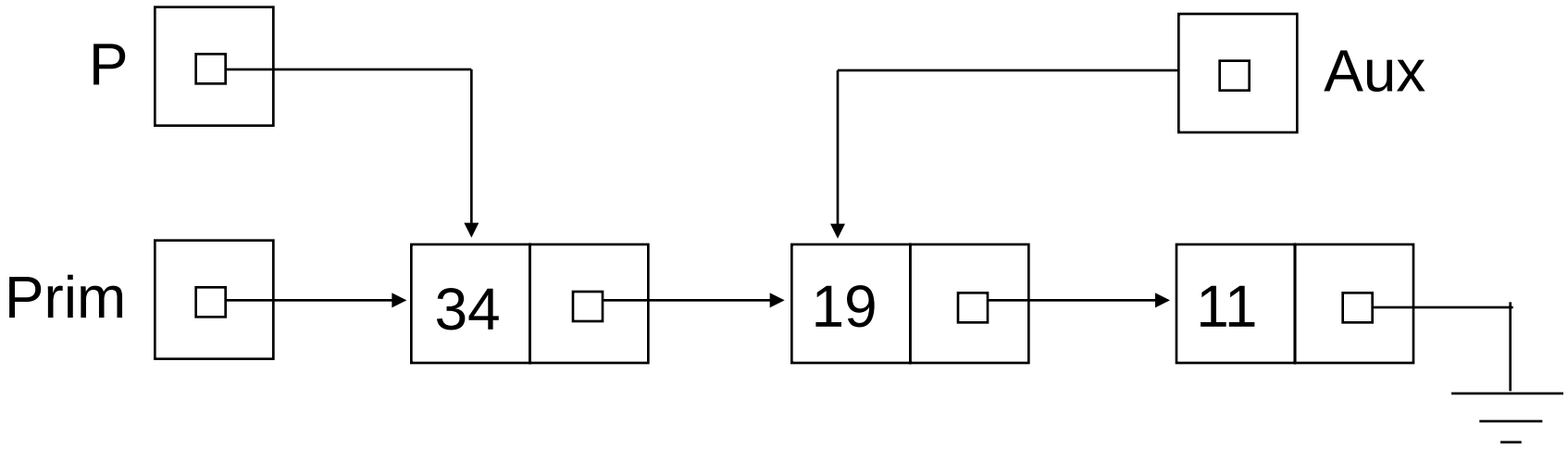
Remoção de um elemento apontado por P.



Listas

Remoção de um elemento apontado por P.

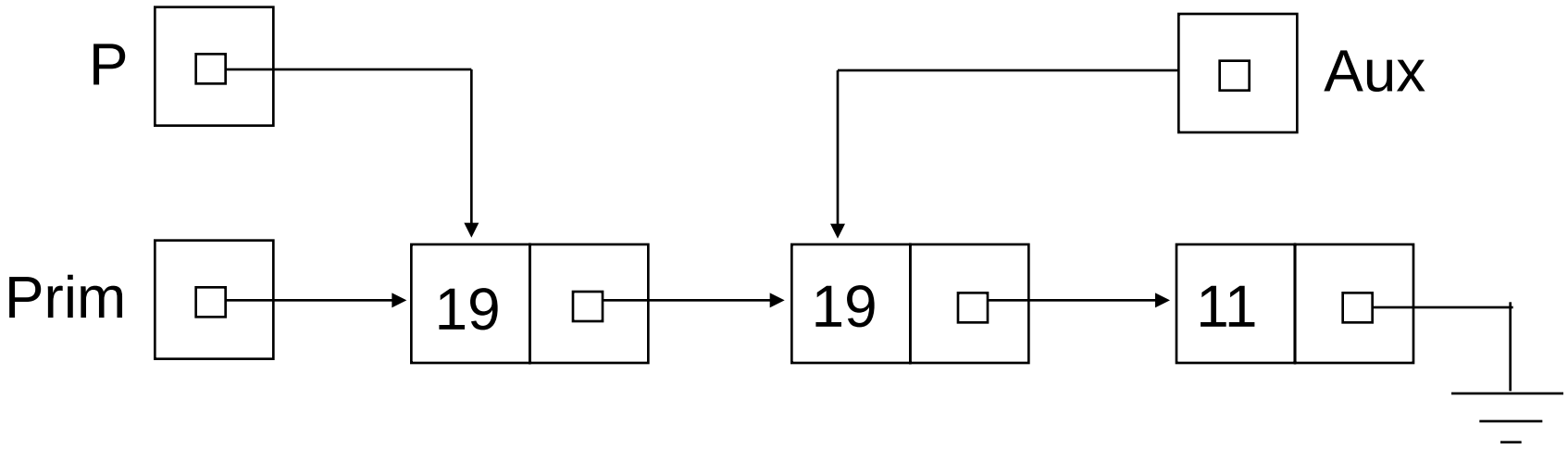
```
Aux := P^.Prox;
```



Listas

Remoção de um elemento apontado por P.

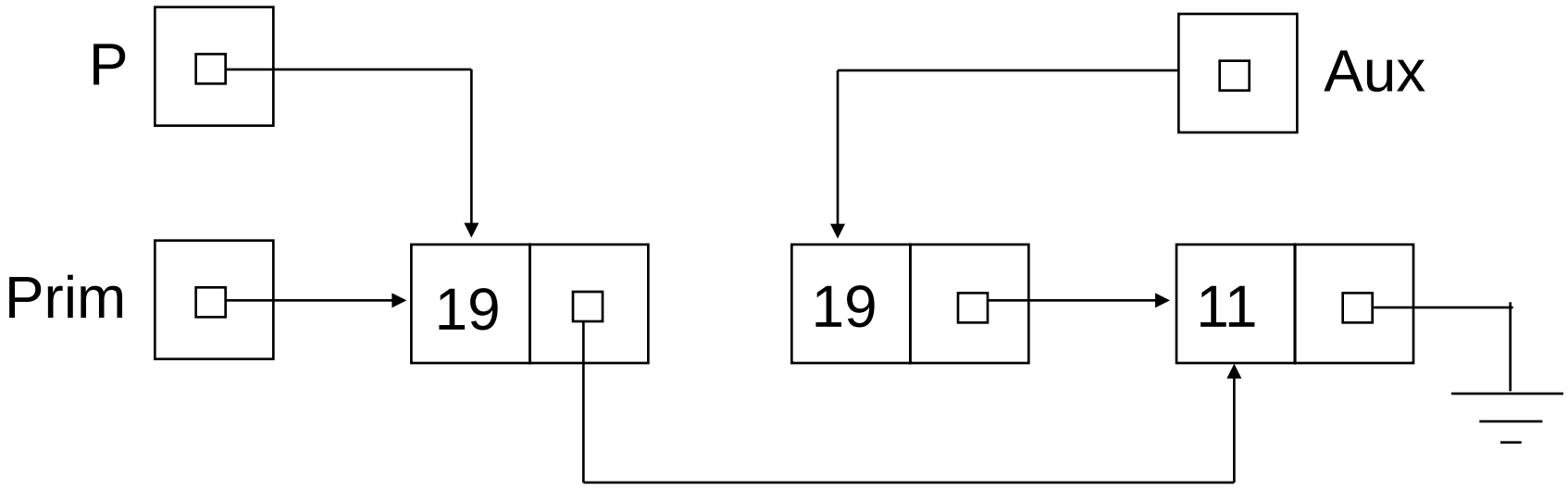
```
P^.Chave := Aux^.Chave;
```



Listas

Remoção de um elemento apontado por P.

```
P^.Prox := Aux^.Prox;
```



Listas

Processamento de todos os elementos da lista.

```
Procedure ProcessaLista (Prim{e} : Ponteiro);
var       P : Ponteiro;
Begin
    P := Prim;
    while (P<>nil) do
        begin
            ProcessaElemento (P);
            P := P^ .Prox
        end;
end;
```

O procedimento acima implementa uma forma geral de acessar e ativar um determinado processamento para todos os elementos da lista.

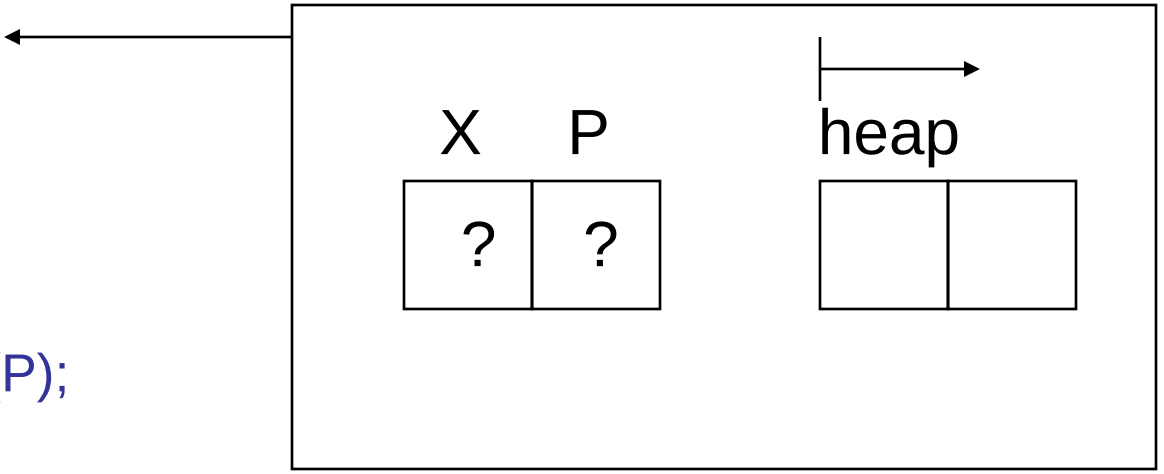
Operador dispose

O operador **dispose**, que recebe um ponteiro **P** como parâmetro, “desliga” a variável dinâmica do ponteiro **P**.

A área de memória ocupada pela variável dinâmica é então liberada de volta ao **heap**.

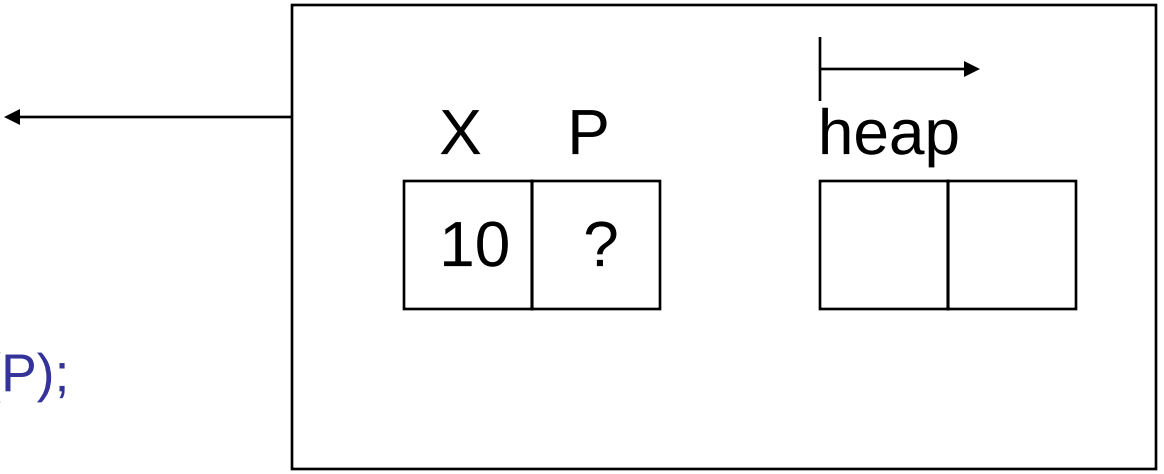
Operador dispose

```
var     X : integer;
        P : ^ integer;
...
X := 10;
new (P);
P^ := X;
...
dispose (P);
...
```



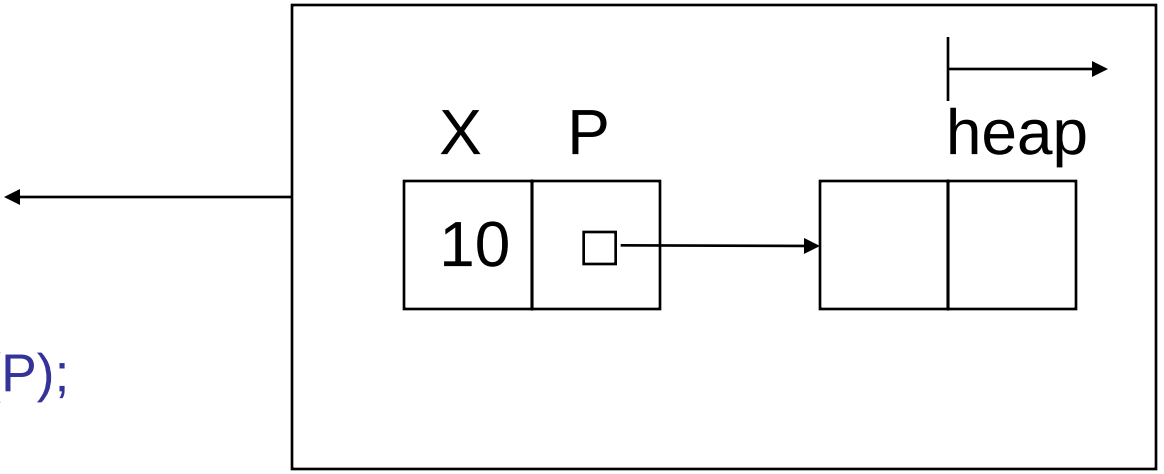
Operador dispose

```
var     X : integer;
        P : ^ integer;
...
X := 10;
new (P);
P^ := X;
...
dispose (P);
...
```



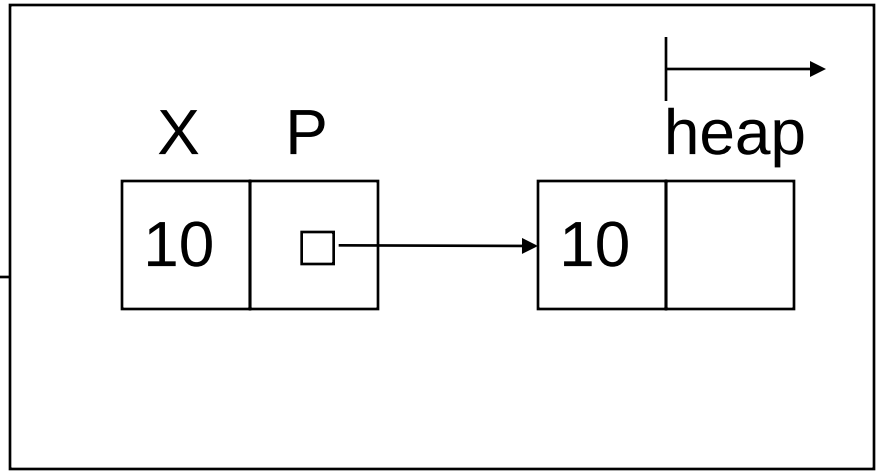
Operador dispose

```
var     X : integer;
        P : ^ integer;
...
X := 10;
new (P);
P^ := X;
...
dispose (P);
...
```



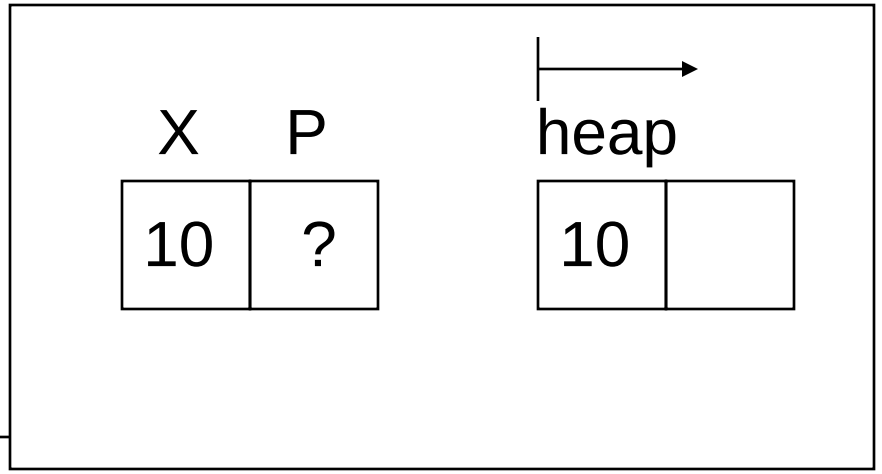
Operador dispose

```
var     X : integer;
        P : ^ integer;
...
X := 10;
new (P);
P^ := X;
...
dispose (P);
...
```



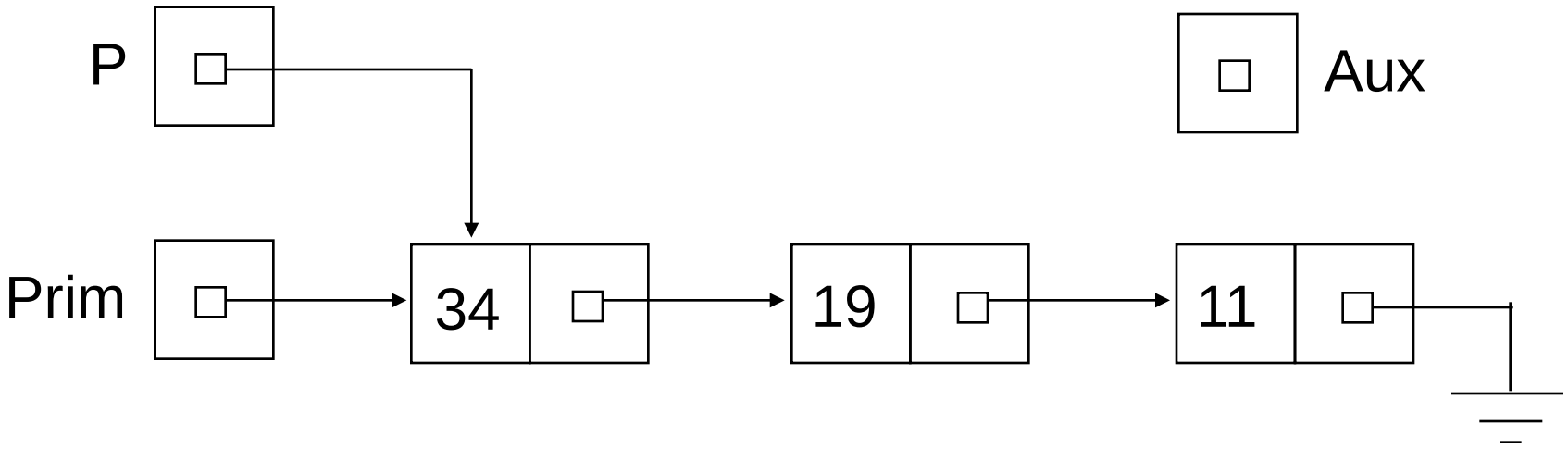
Operador dispose

```
var     X : integer;
        P : ^ integer;
...
X := 10;
new (P);
P^ := X;
...
dispose (P);
...
```



Operador dispose

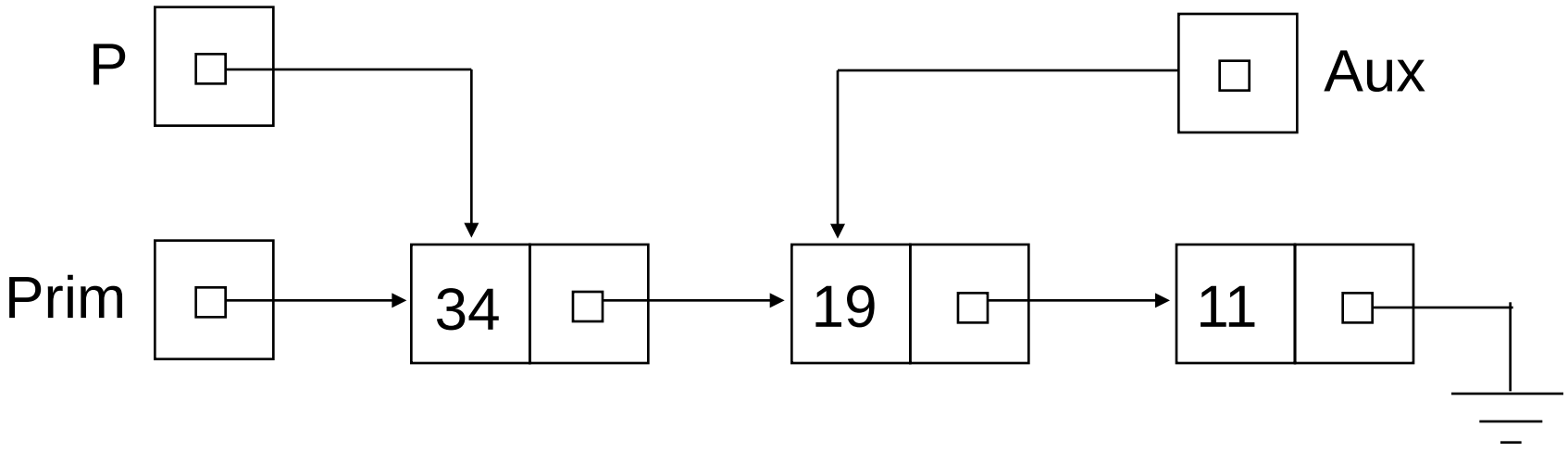
Revendo os passos da remoção de um elemento apontado por P.



Operador dispose

Revendo os passos da remoção de um elemento apontado por P.

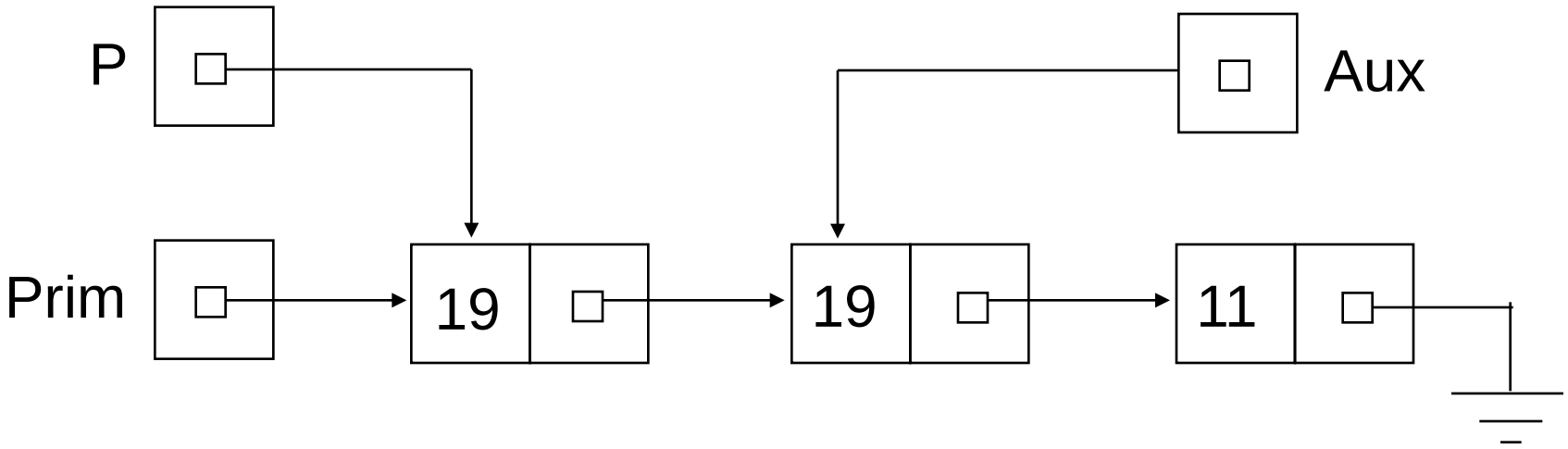
Aux := P^.Prox;



Listas

Revendo os passos da remoção de um elemento apontado por P.

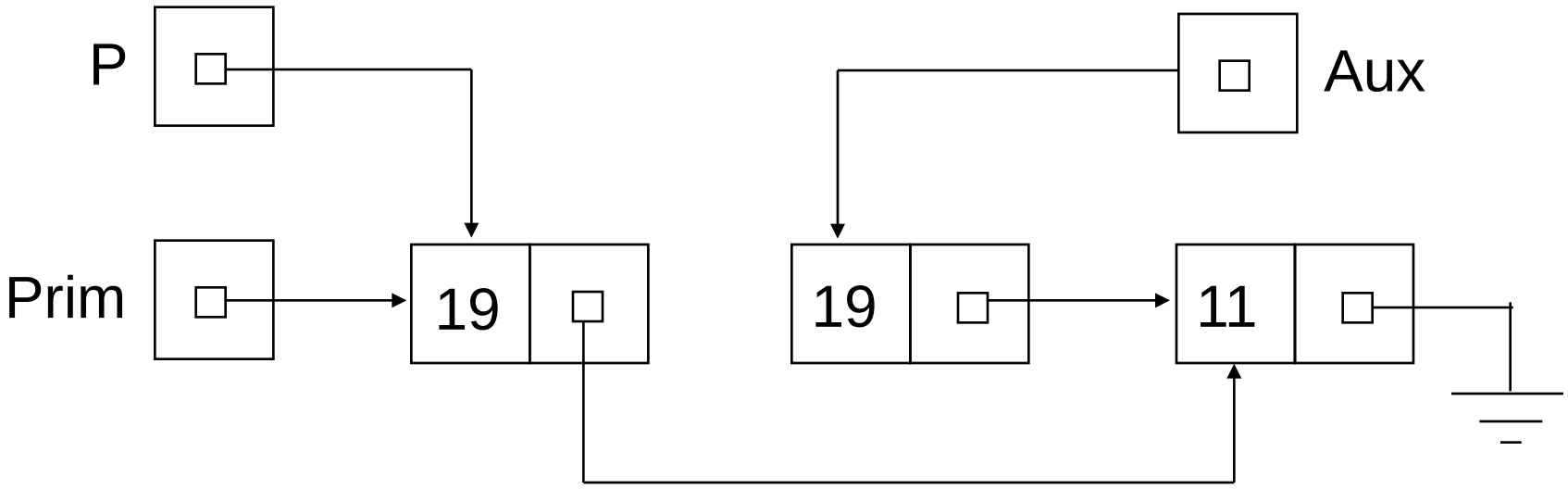
P^.Chave := Aux^.Chave;



Listas

Revendo os passos da remoção de um elemento apontado por P.

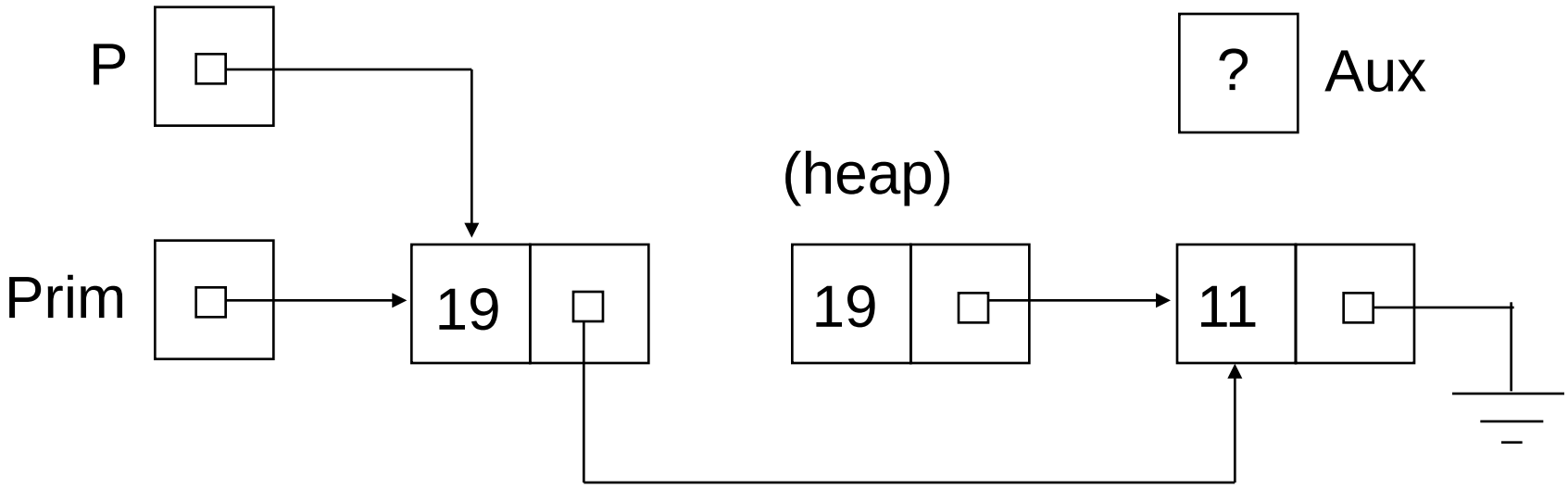
P^.Prox := Aux^.Prox;



Listas

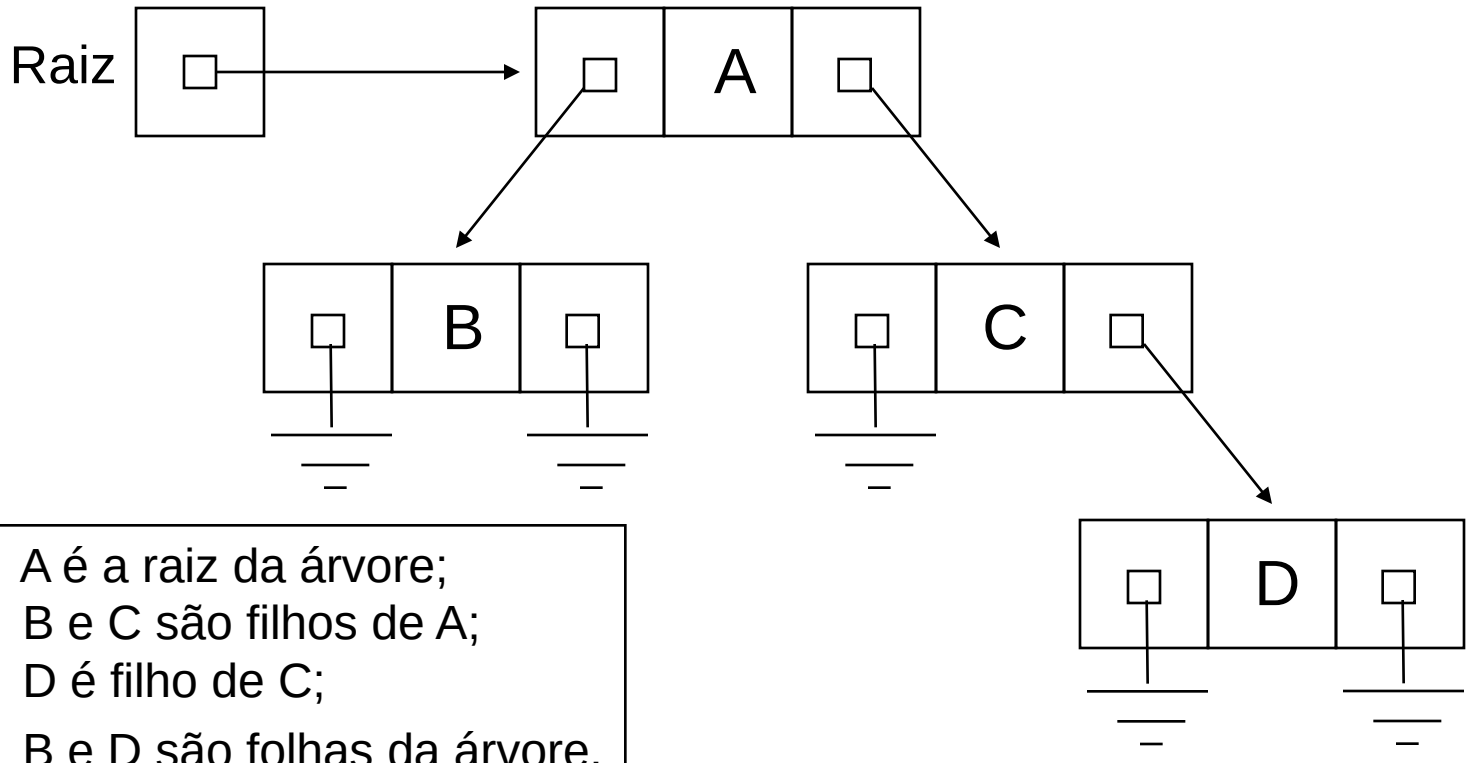
Revendo os passos da remoção de um elemento apontado por P.

Dispose (Aux);



Árvores

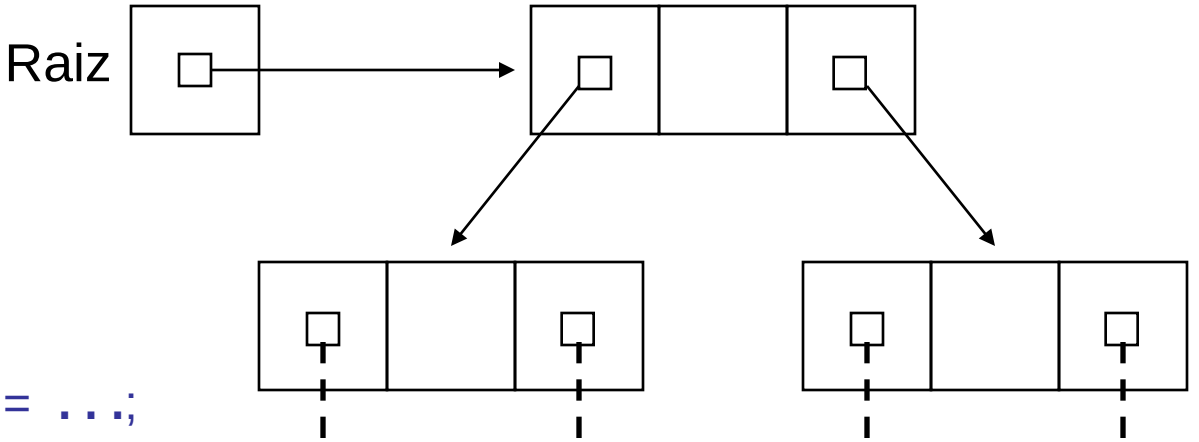
Árvores também são estruturas de dados comumente implementadas em Pascal utilizando-se ponteiros.



- A é a raiz da árvore;
- B e C são filhos de A;
- D é filho de C;
- B e D são folhas da árvore.

Árvores

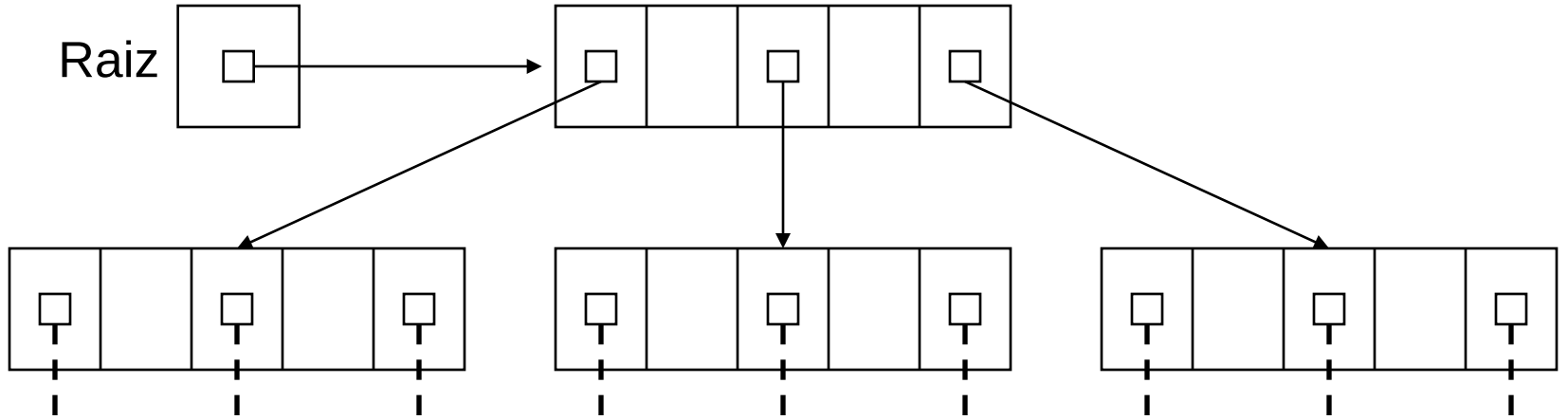
Estrutura de dados recursiva que define uma árvore binária.




```
type    Informacao = . . .;
        Filho = ^ Elemento;
        Elemento = record
                     Esquerdo : Filho;
                     Chave : Informacao;
                     Direito : Filho
                   end;

var     Raiz : Filho;
```

Árvores

Estrutura de dados recursiva que define uma árvore ternária.




```
type    Informacao = . . .;                    var    Raiz : Filho;
        Filho = ^ Elemento;
        Elemento = record
                     F1 : Filho;
                     Info1 : Informacao;
                     F2 : Filho;
                     Info2 : Informacao;
                     F3 : Filho;
                   end;
```

Aula 11-A

Conteúdo

Ponteiros (Pointers)