

Haskell

- Linguagem de programação de uso genérico
- Funcional
- Estaticamente (fortemente) tipada
- Interpretada e compilada
- Alto nível
- Criada em 1990 por um comité

C

- Linguagem de programação de uso genérico
- Imperativa e procedural
- Estaticamente (fracamente) tipada
- Compilada
- Baixo nível
- Criada em ??? por Dennis Ritchie

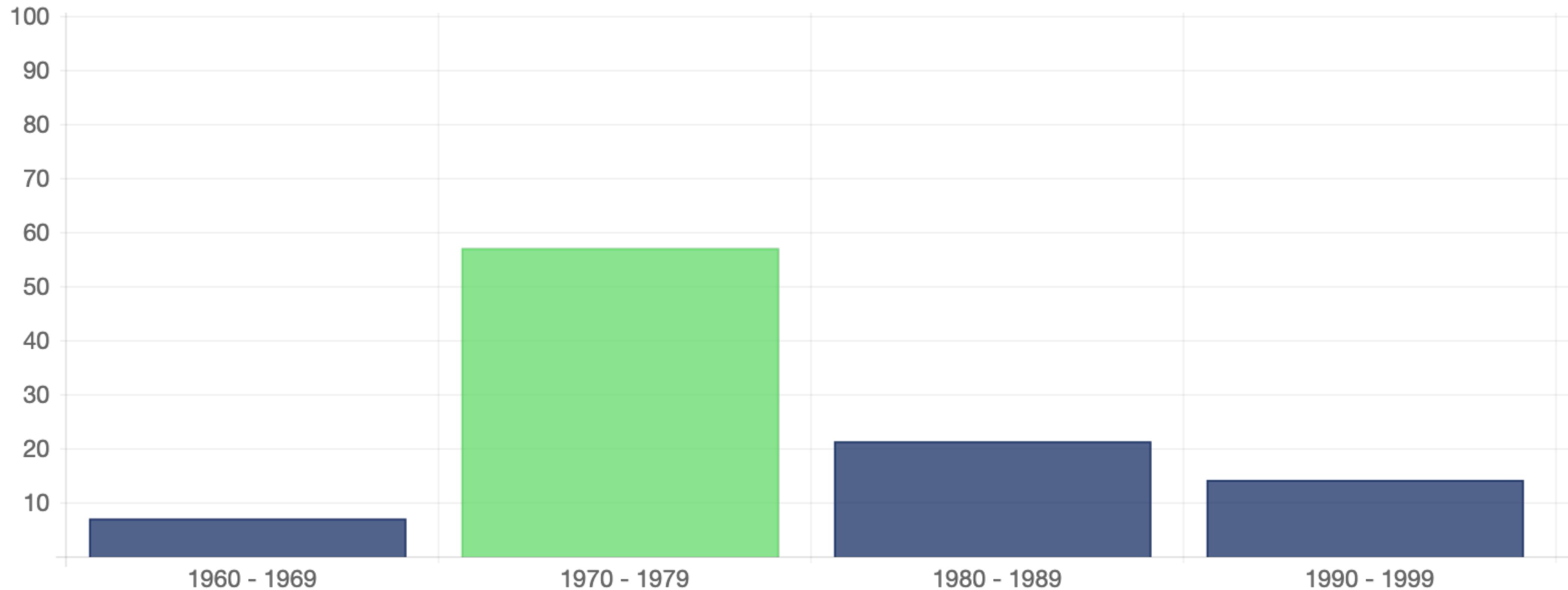


#0 Em que década foi criada a linguagem C?

- 1960 - 1969
- 1970 - 1979
- 1980 - 1989
- 1990 - 1999

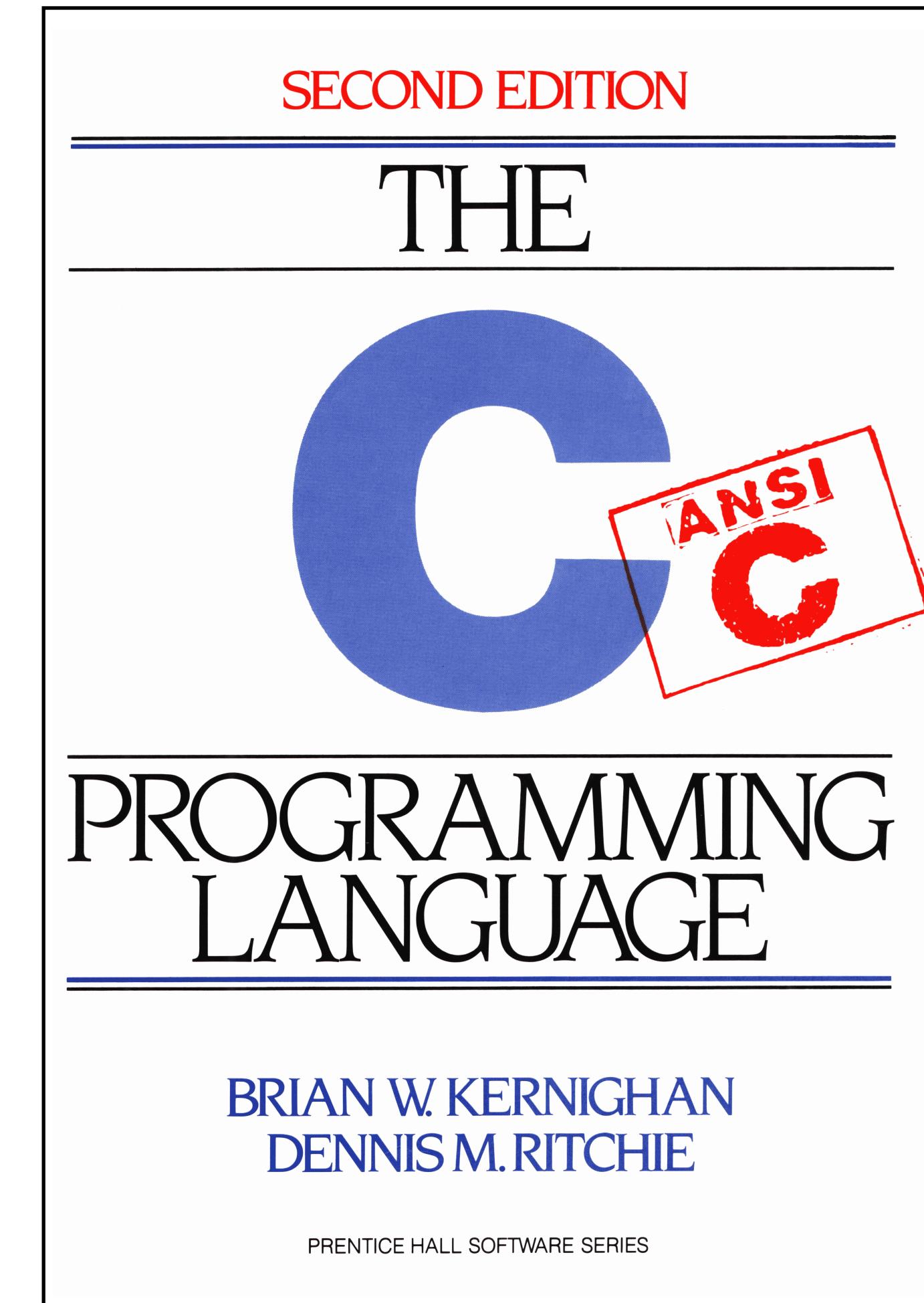


#0 Em que década foi criada a linguagem C?

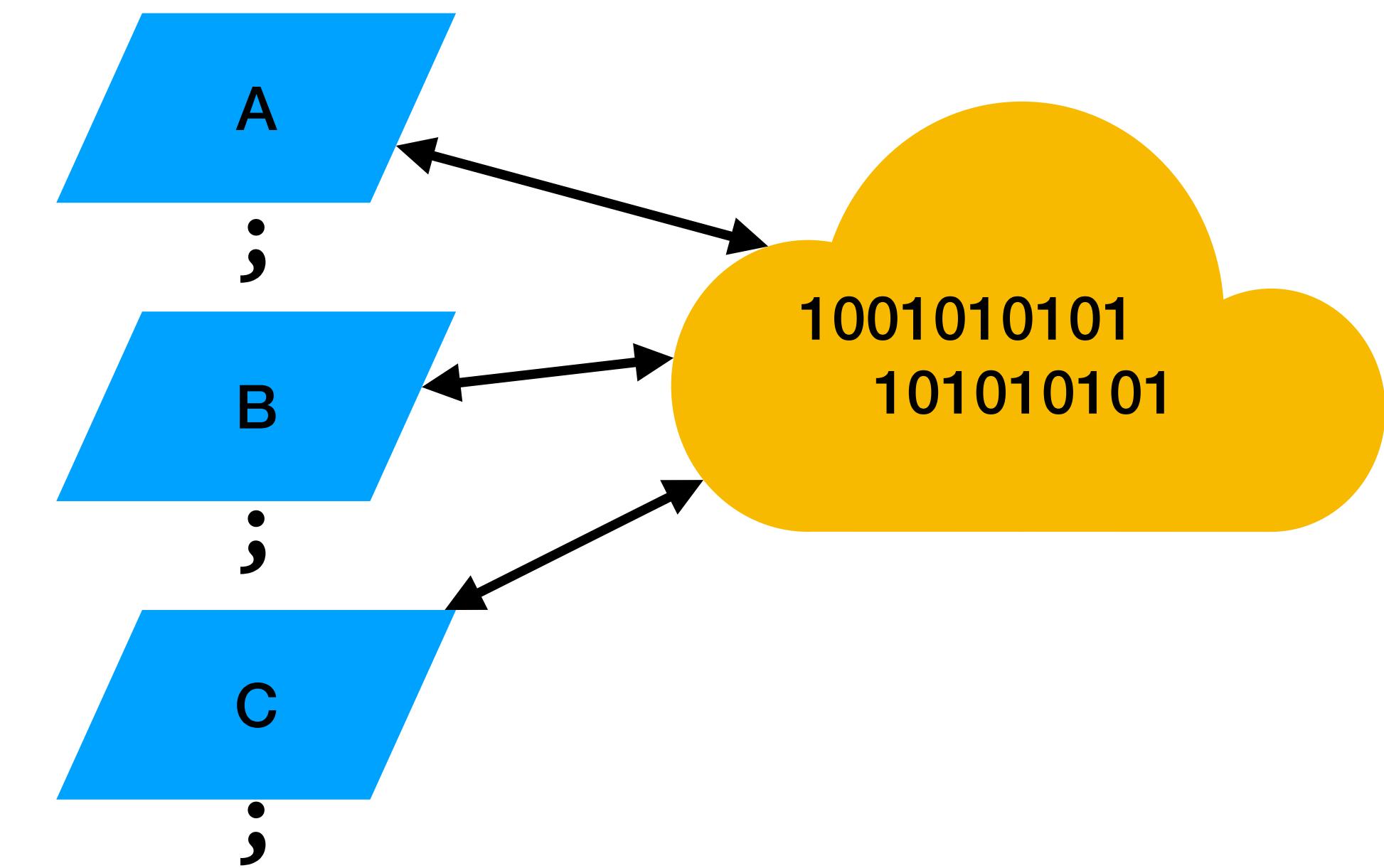
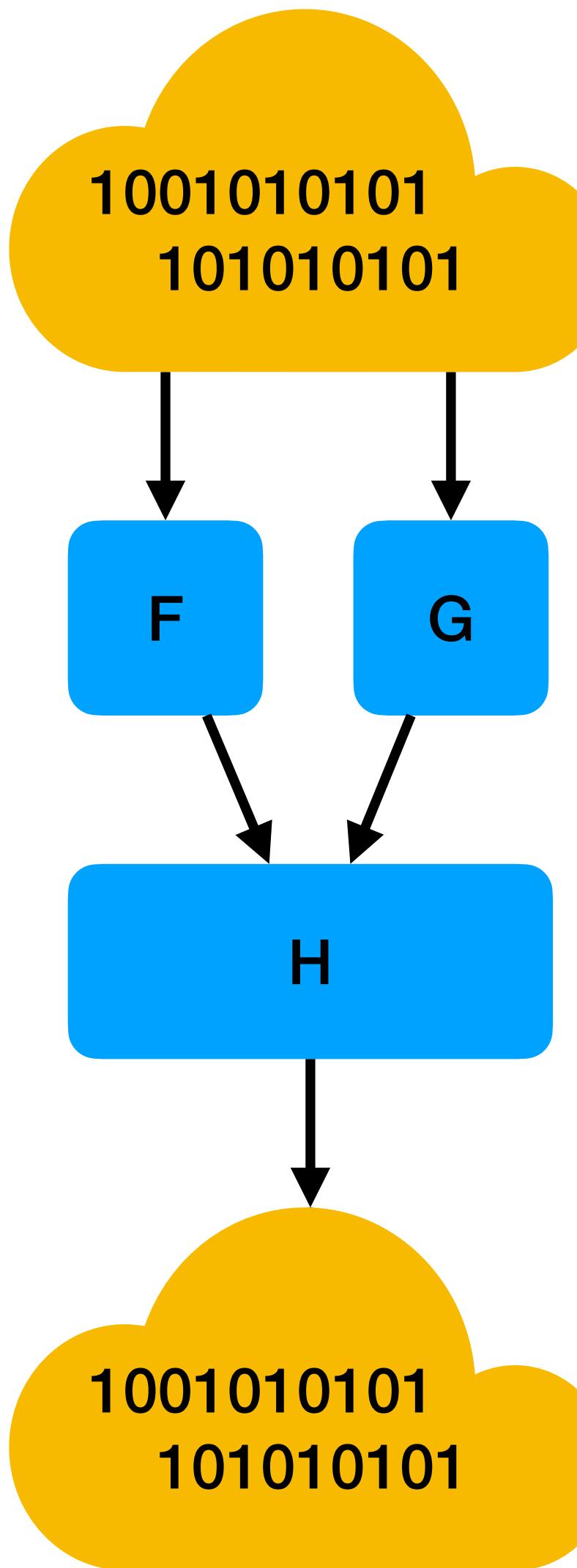


História e Bibliografia

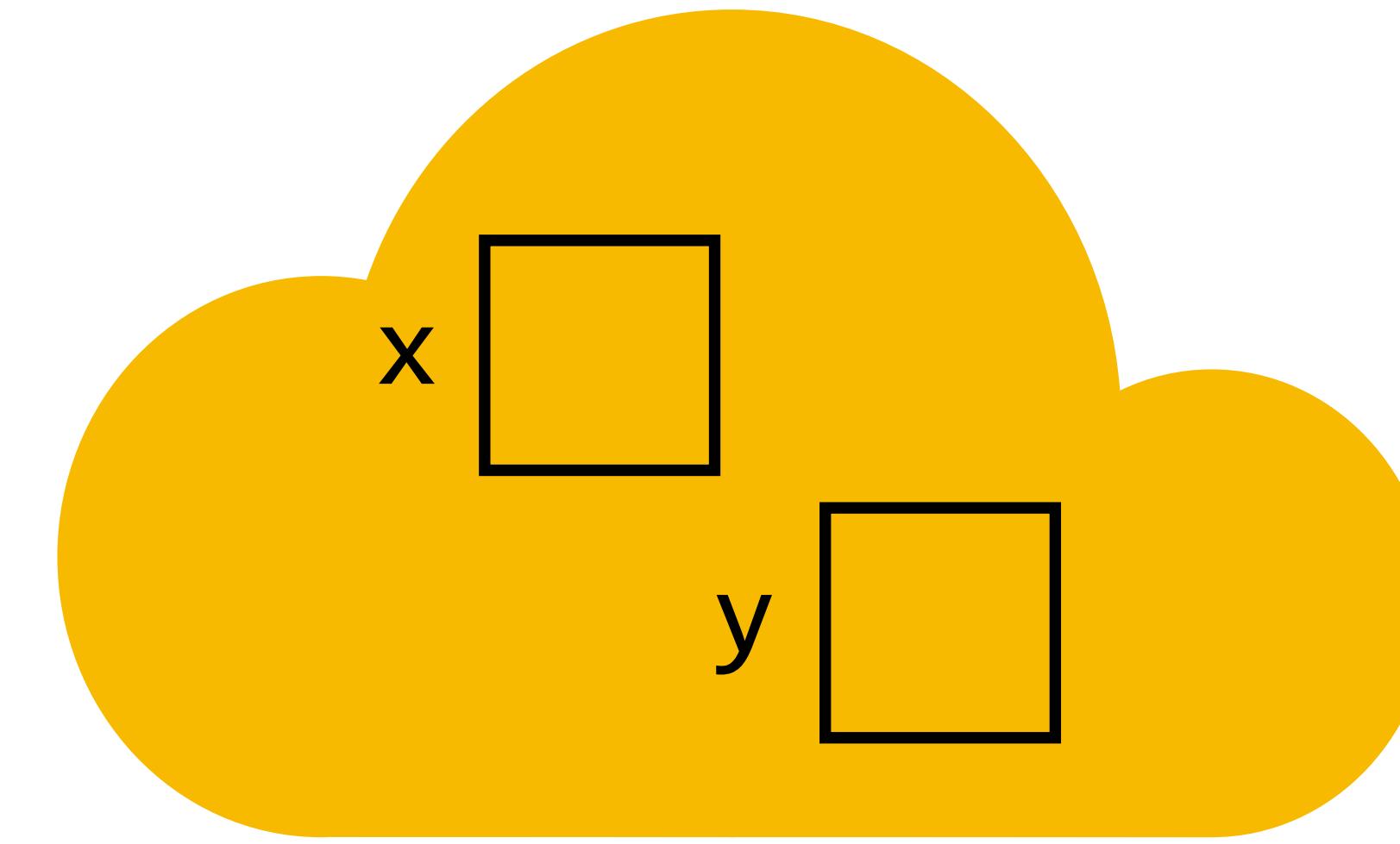
- C (1972)
- K&R C (1978)
- ANSI C (1989)
- C99 (1999)
- C11 (2011)
- ...



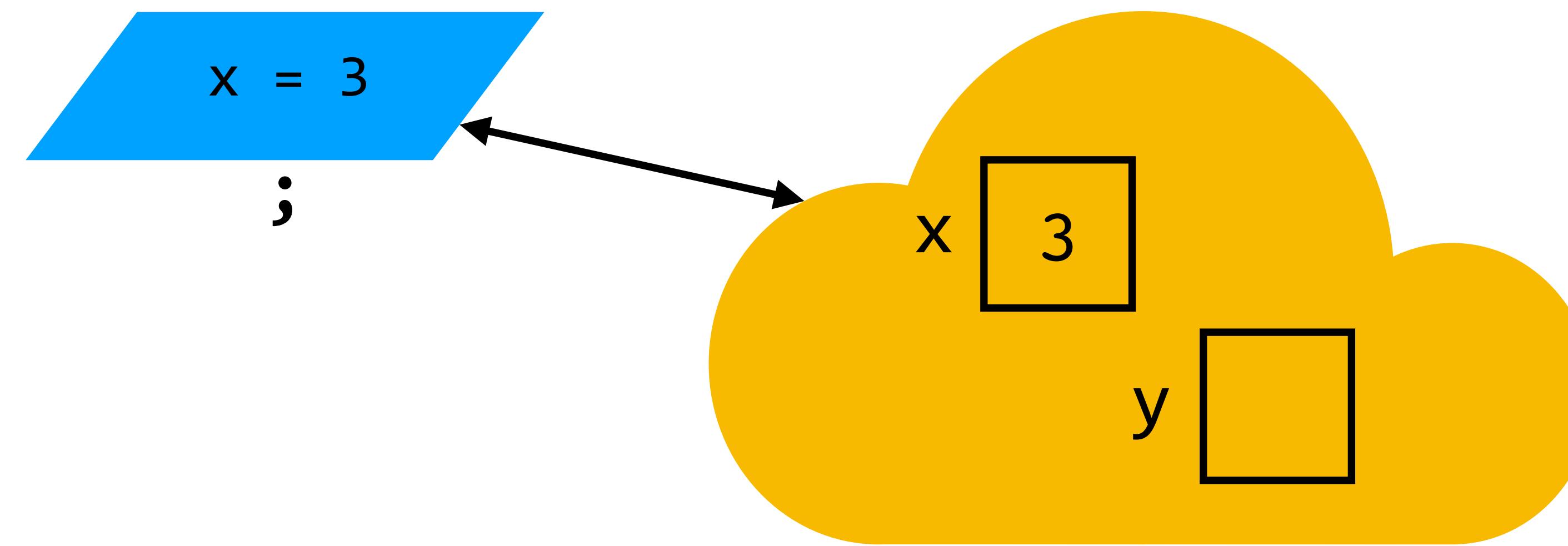
Funcional vs Imperativo



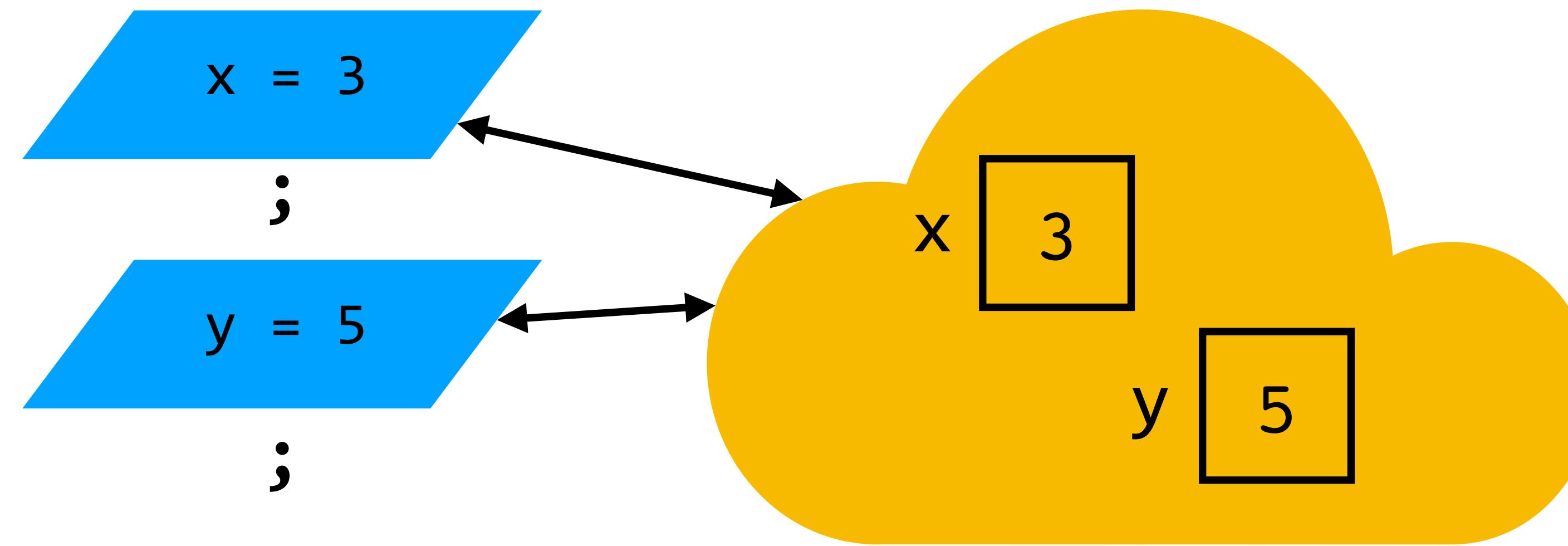
Variáveis e Atribuição



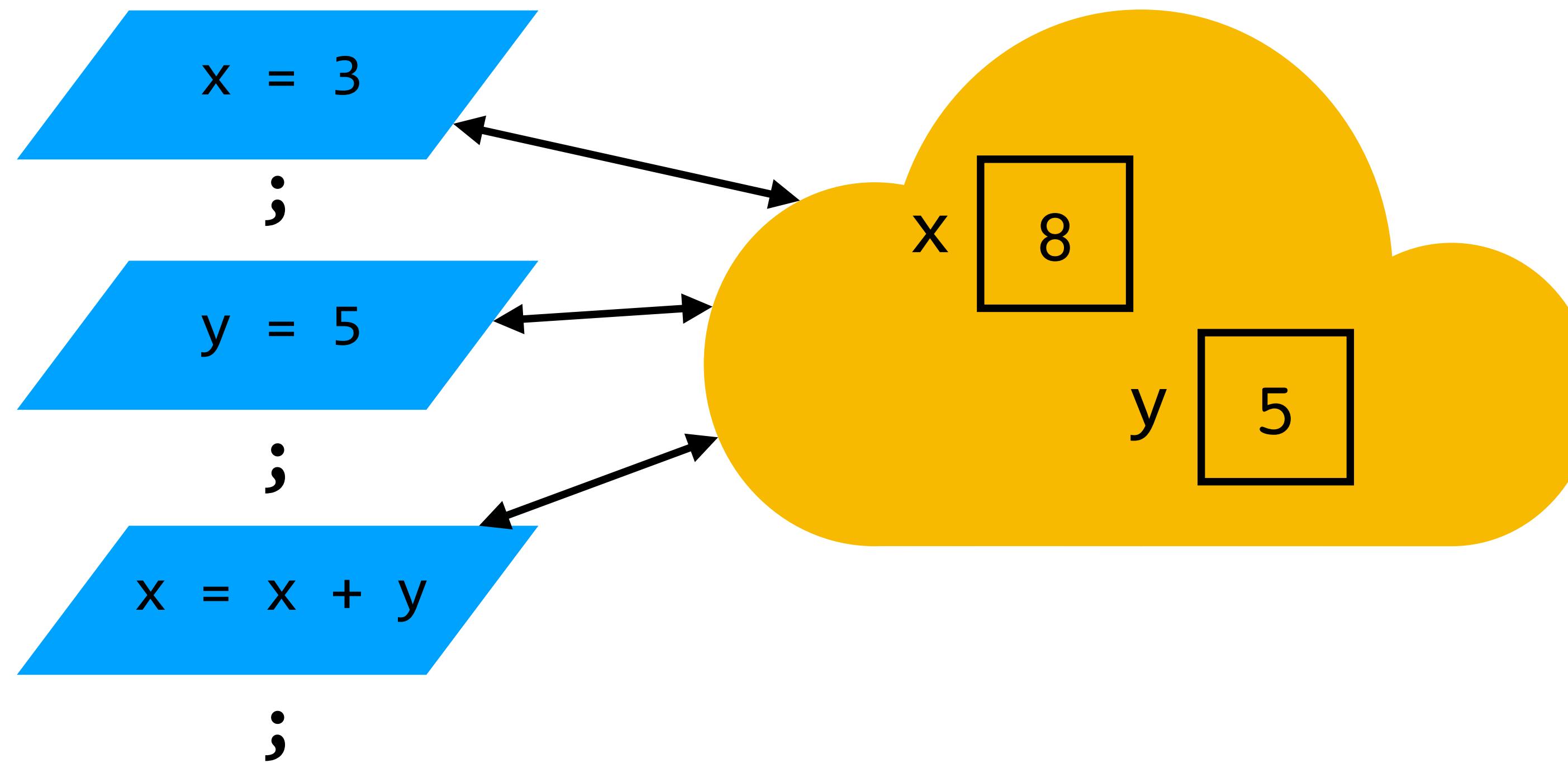
Variáveis e Atribuição



Variáveis e Atribuição



Variáveis e Atribuição



Programas

- Um *programa C* é uma sequência de declarações de *variáveis*, *tipos* ou *funções*, definições de funções, e *directivas de pré-processamento*
- Uma variável, tipo ou função só pode ser usada se declarada antes
- O espaçamento e indentação não têm significado
- As declarações são terminadas por ponto e vírgula
- Um programa pode estar dividido em vários ficheiros (aka módulos ou bibliotecas)
- A execução de um programa começa na função especial `main`

Declaração de variáveis

tipo id;

tipo id = expr;

tipo id₀, id₁, ...;

tipo id₀ = expr₀, id₁ = expr₀, ...;

Tipos numéricos

Tipo	Modificadores	Tamanho
char	signed, unsigned	≥ 1
short	signed (default), unsigned	≥ 2
int	signed (default), unsigned	≥ 2
long	signed (default), unsigned	≥ 4
float		precisão simples
double		precisão dupla

Funções

- A *declaração* de uma função inclui os tipos dos parâmetros e o tipo do valor retornado
- Uma função pode não ter parâmetros, nem retornar qualquer valor, usando-se neste caso o tipo especial **void** como tipo de retorno
- A declaração de uma função é normalmente seguida da respectiva *definição*, mas podem estar separadas
- A definição de uma função é uma sequência declarações (de variáveis ou tipos) e de *comandos* entre chavetas
- Tal como as declarações, os comandos são terminados por ponto e vírgula
- O comando **return** é usado para retornar um valor

Definição de funções

```
tipo id (tipo1 id1, ..., tipon idn) {  
    // declarações e comandos  
return expr;  
}
```

Exemplo

```
int dobro(int a) {  
    int r;  
    r = 2*a;  
    return r;  
}
```

```
int main() {  
    int r;  
    r = dobro(3);  
    return 0;  
}
```

Compilação

```
int dobro(int a) {  
    int r;  
    r = 2*a;  
    return r;  
}  
  
int main() {  
    int r;  
    r = dobro(3);  
    return 0;  
}
```

prog.c



A screenshot of a terminal window titled "alcino — -zsh — 50x10". The window shows the following command-line session:

```
alcino@Nausicaa ~ % gcc prog.c  
alcino@Nausicaa ~ % ./a.out  
[alcino@Nausicaa ~ % gcc -o prog prog.c  
[alcino@Nausicaa ~ % ./prog  
alcino@Nausicaa ~ % ]
```

Aula 2

Declaração vs Definição

```
int dobro(int);           // Declaração
```

```
int main() {  
    int r;  
    r = dobro(3);  
    return 0;  
}
```

```
int dobro(int a) { // Definição  
    int r;  
    r = 2*a;  
    return r;  
}
```

Directiva #include

```
int dobro(int a) {  
    int r;  
    r = 2*a;  
    return r;  
}
```

dobro.c

```
#include "dobro.c"  
  
int main() {  
    int r;  
    r = dobro(3);  
    return 0;  
}
```

prog.c

Bibliotecas pré-definidas

Nome	Conteúdo
<stdio.h>	Input e output
<stdlib.h>	Conversão de tipos, geração de números aleatórios, alocação de memória, ...
<math.h>	Funções matemáticas (exponenciação, raíz quadrada, trigonométricas, ...)
<string.h>	Manipulação de strings

A função printf

```
#include <stdio.h>

int main() {
    printf("Olá mundo!\n");
    printf("O dobro de %s é %d.\n", "dois", 4);
    return 0;
}
```

Códigos de formatação

Código	Formatação
%d	Número inteiro em notação decimal com sinal
%x	Número inteiro em notação hexadecimal
%f	Número real em notação decimal
%e	Número real em notação científica
%c	Caracter
%s	String
%%	Caracter '%'

Expressões e Comandos

- As *expressões* denotam um valor
 - Definidas à custa de variáveis, constantes, operadores e funções
- Os *comandos* afectam o estado do programa
 - Uma atribuição é um comando

Expressões aritméticas

Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão inteira

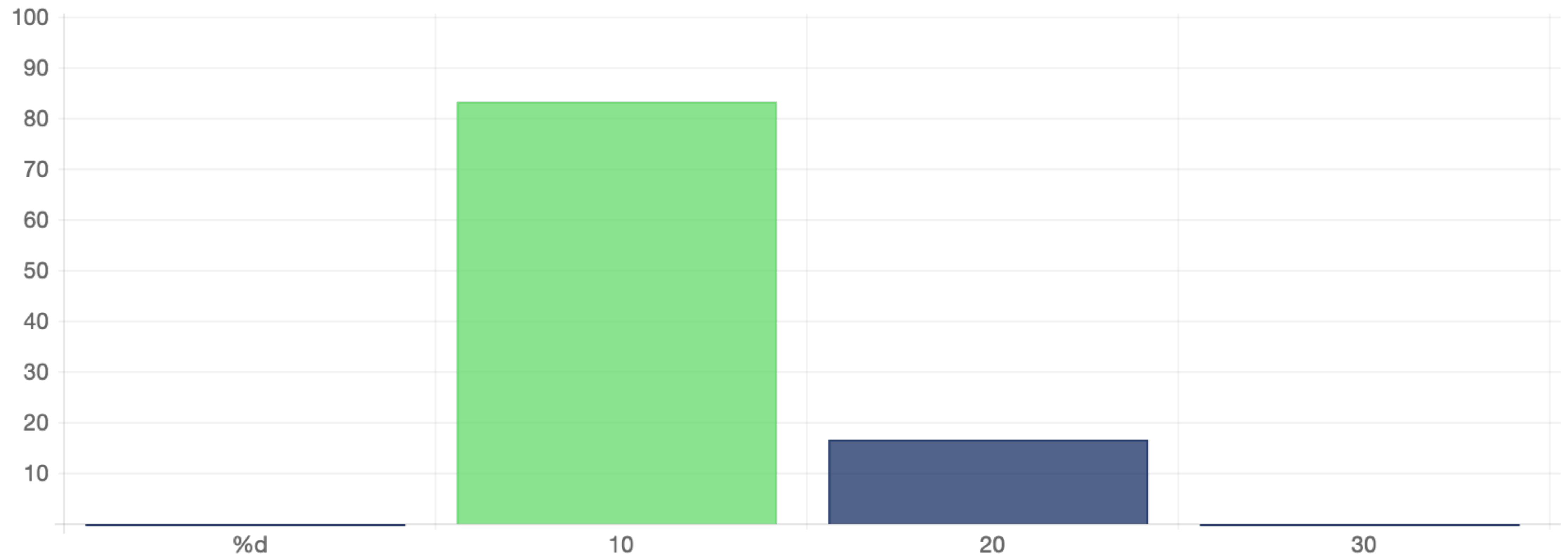
#1 Que imprime o seguinte programa?

```
#include <stdio.h>

int main() {
    int x = 20, y = 10;
    x = x + y;
    y = x - y;
    x = x - y;
    printf("%d\n", x);
    return 0;
}
```



#1 Que imprime o seguinte programa?



<https://pythontutor.com/c.html>

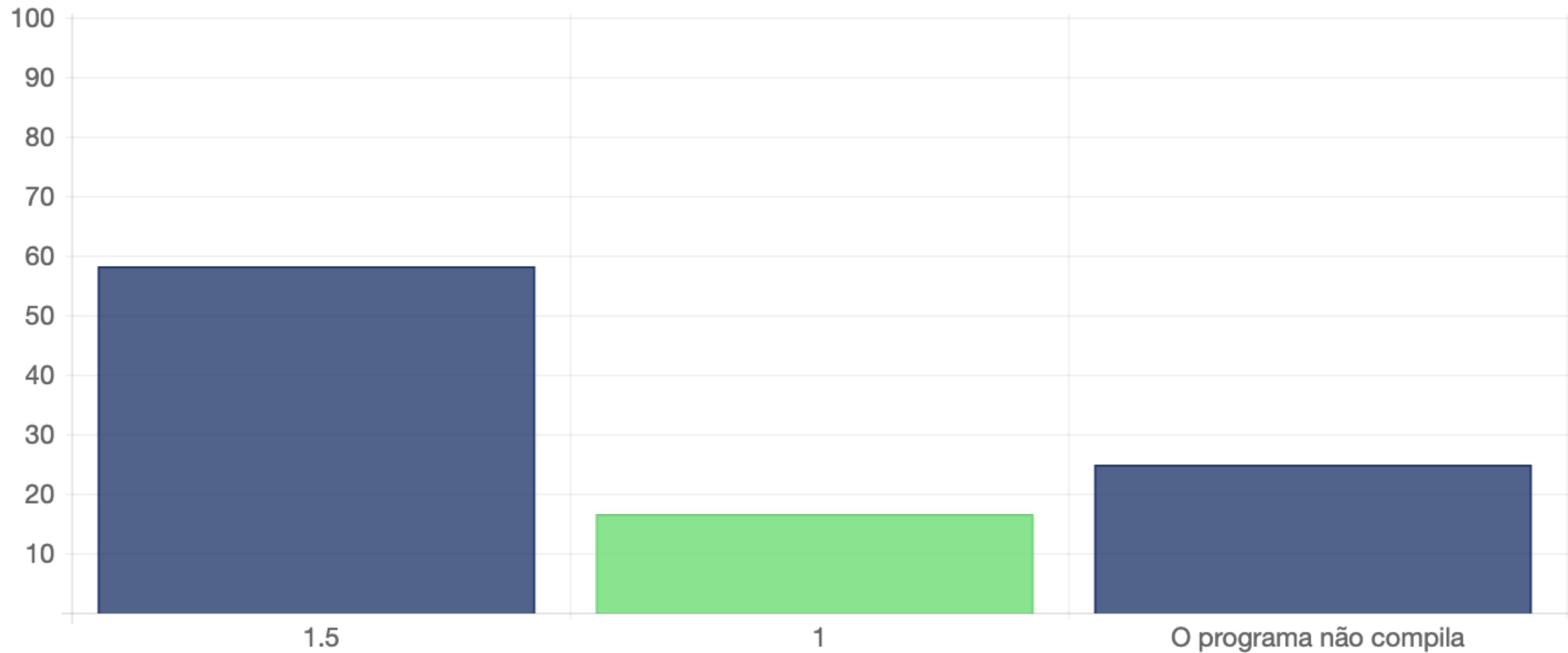


#2 Qual o valor de f depois da atribuição?

```
int main() {  
    float f;  
    f = 3 / 2;  
    return 0;  
}
```



#2 Qual o valor de f depois da atribuição?



Conversões entre tipos

- Numa operação aritmética o operando de tipo “mais pequeno” é convertido para o tipo “maior”, sendo a operação realizada no tipo “maior”
 - Se um operando é um **double** o outro é convertido para **double**
 - Senão, se um operando é um **float** o outro é convertido para **float**
 - Senão, qualquer **char** ou **short** é promovido para **int**
 - Depois, se um operando é **long** o outro é convertido para **long**
- Numa atribuição o valor da expressão é convertido implicitamente para o tipo da variável
 - Uma conversão pode também ser feita de forma explícita com o operador unário (*tipo*)

Conversões entre tipos



Divisão exacta

```
int main() {  
    float f;  
    f = 3 / 2.0;  
    return 0;  
}
```

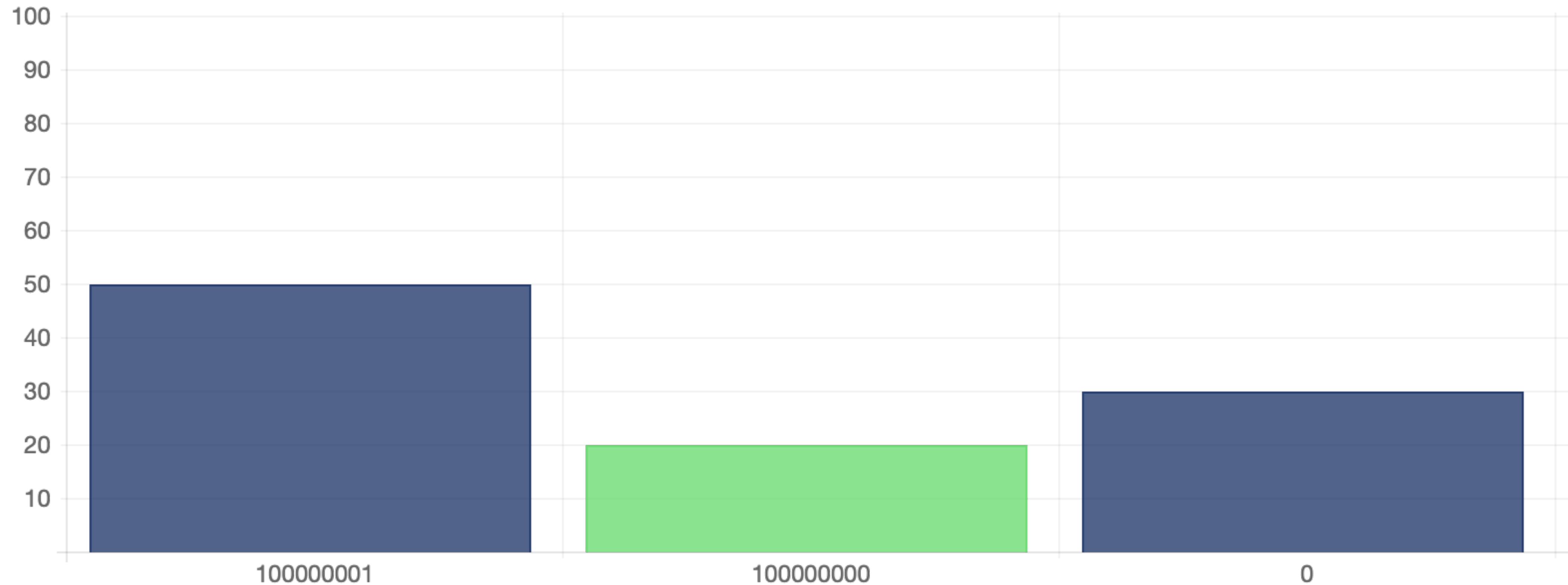
```
int main() {  
    float f;  
    f = 3 / (float) 2;  
    return 0;  
}
```

#3 Qual o valor de x no final?

```
int main() {  
    int x = 100000001;  
    float f;  
    f = x;  
    x = f;  
    return 0;  
};
```



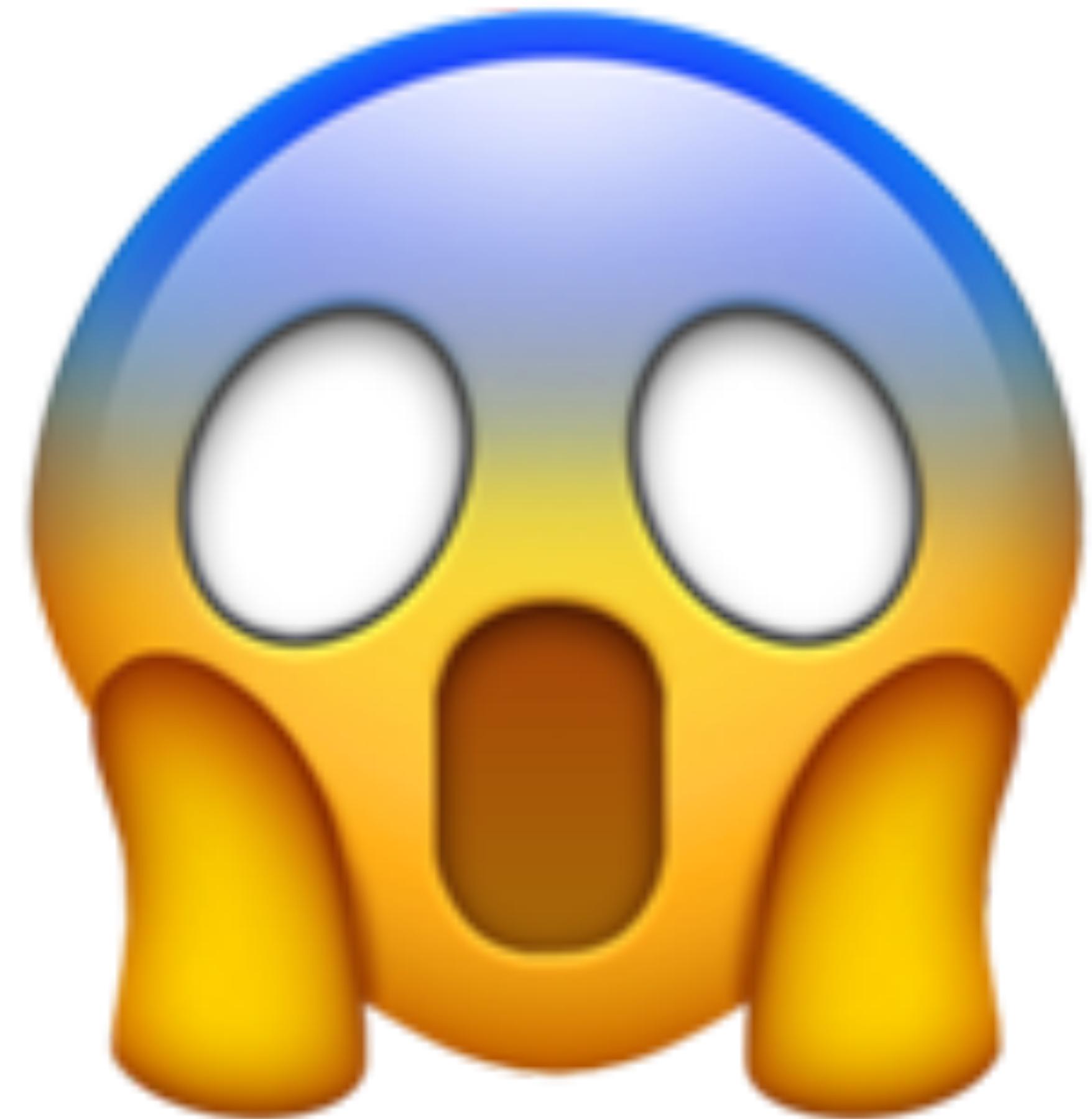
#3 Qual o valor de x no final?



Máximo

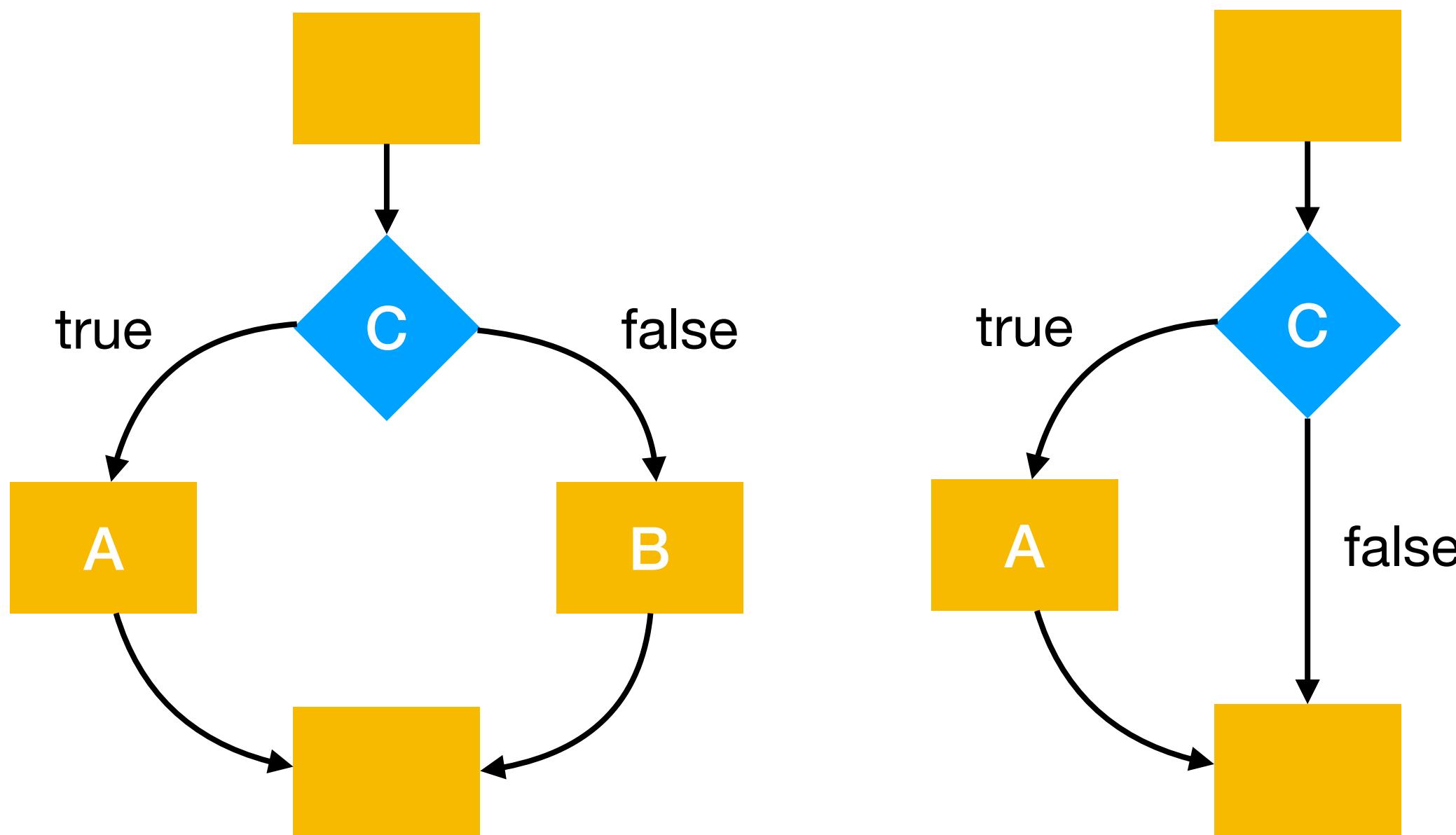
```
#include <stdlib.h>

int max(int a, int b) {
    int r;
    r = (a + b + abs(a-b)) / 2;
    return r;
}
```



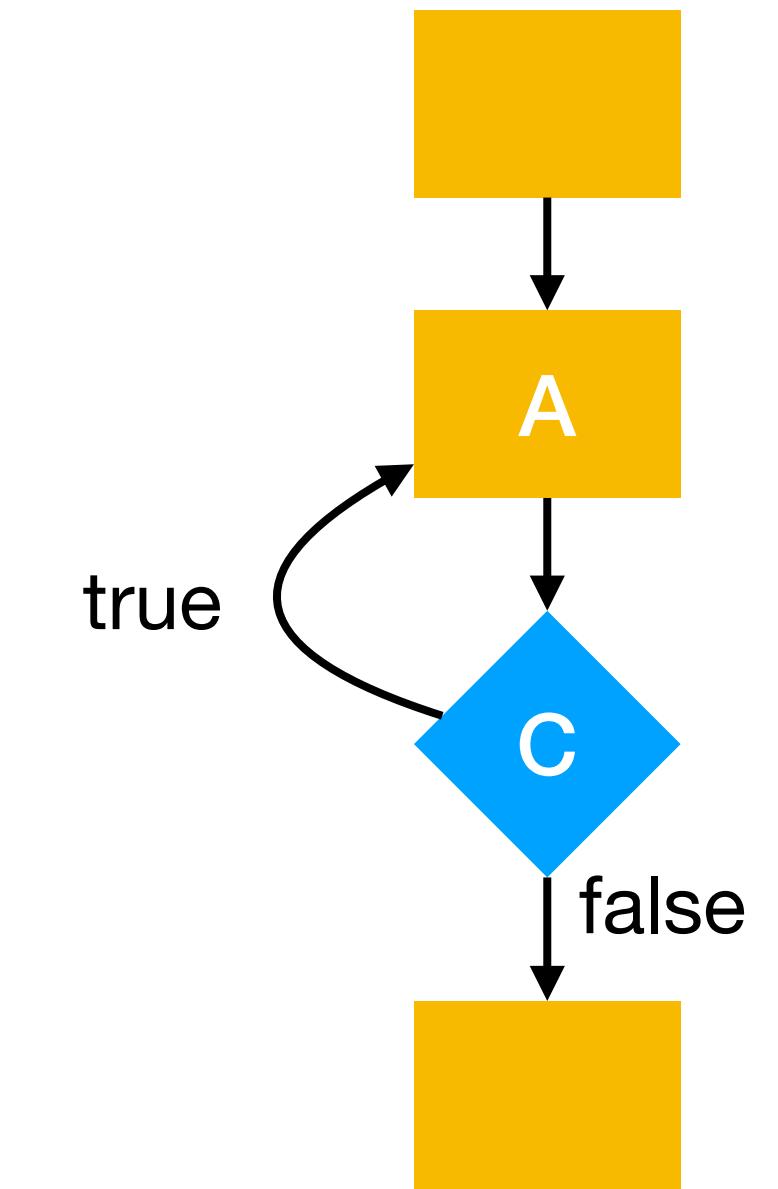
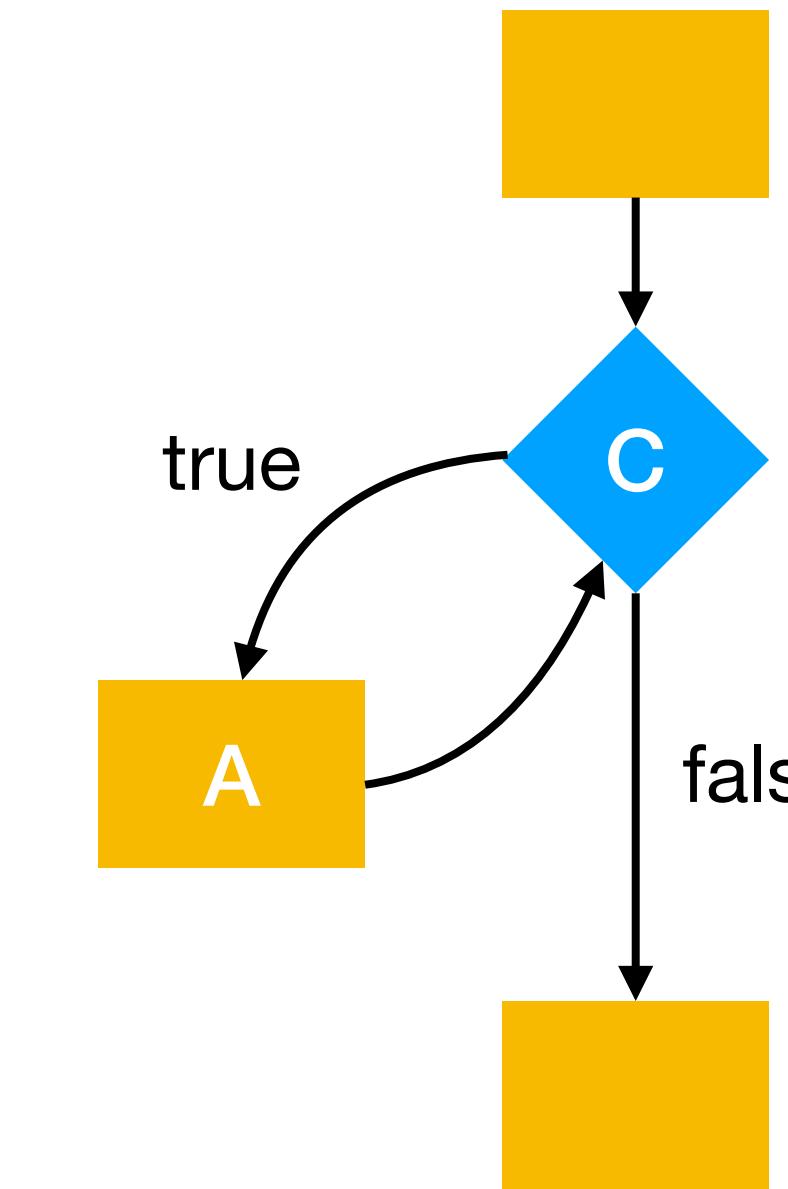
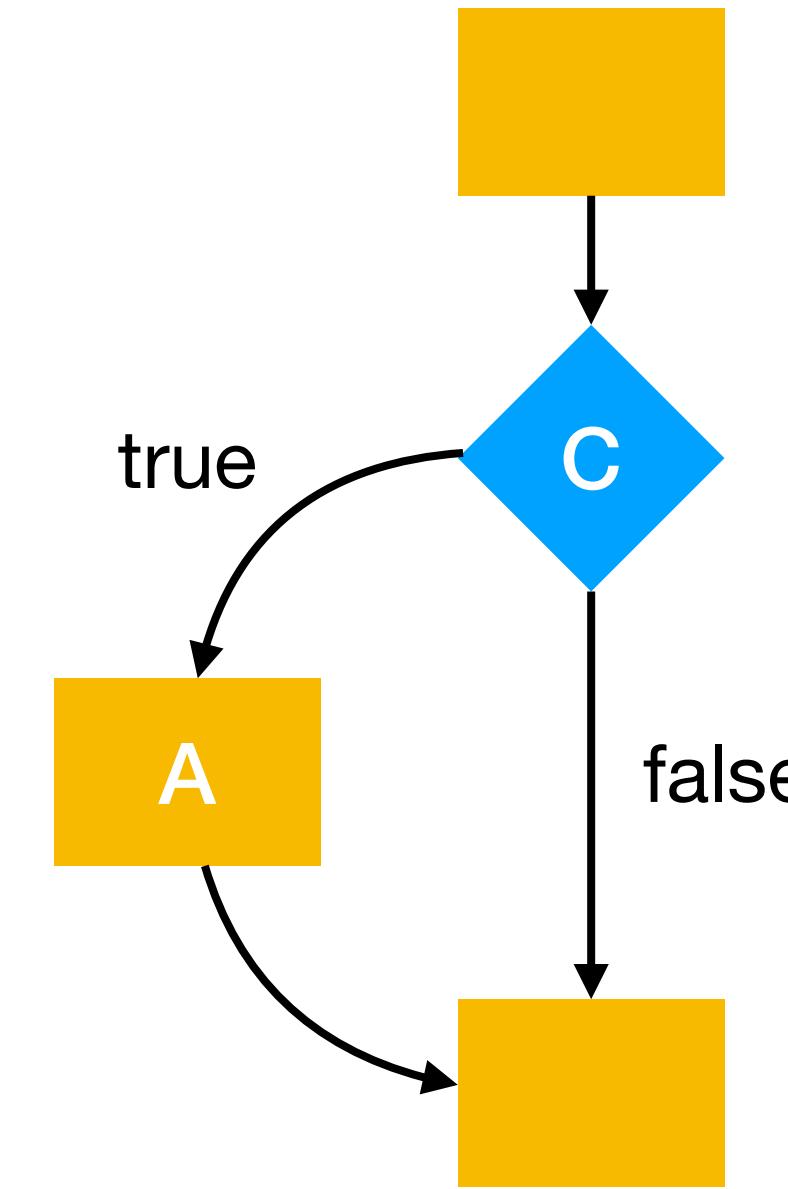
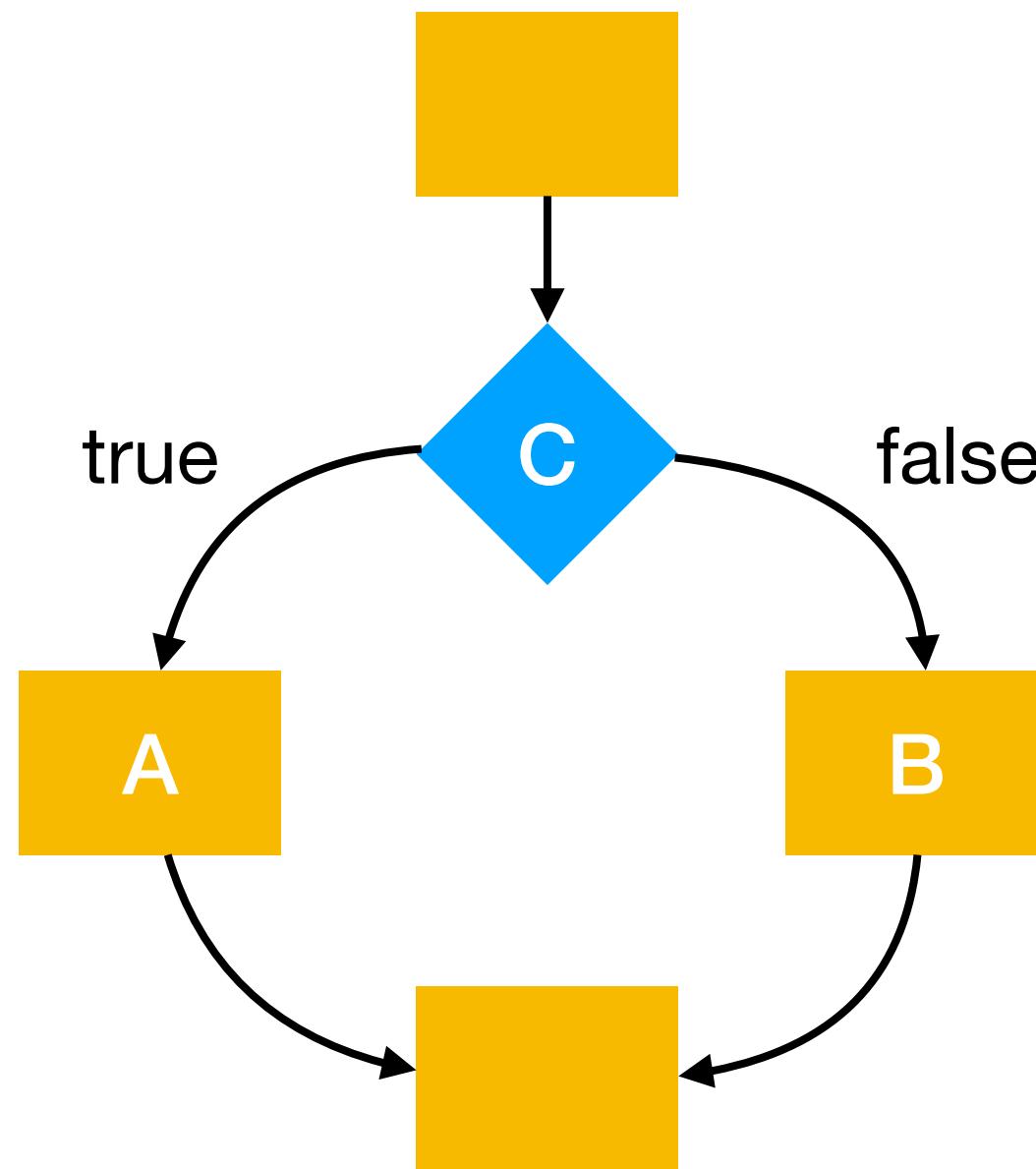
Comandos de controlo

Comandos de controlo



Condicionais

Comandos de controlo



Condicionais

Ciclos

Condicional

if (*expr*) *cmd₁*; **else** *cmd₂*;

if (*expr*) *cmd*;

Operadores relacionais

Operador	Significado
<code>==</code>	Igualdade
<code>!=</code>	Diferença
<code><</code>	Menor
<code><=</code>	Menor ou igual
<code>></code>	Maior
<code><=</code>	Menor ou igual

Operadores lógicos

Operador	Significado
!	Negação
&&	Conjunção
	Disjunção

Booleanos

- Não existe o tipo booleano em C
- Qualquer expressão numérica pode ser usada como booleano
 - O valor 0 corresponde a falso
 - Um valor diferente de 0 corresponde a verdadeiro
- Os operadores relacionais e lógicos devolvem 0 quando o resultado é falso e 1 quando é verdadeiro

Aula 3

Máximo

```
int max(int a, int b) {  
    int r;  
    if (a > b) r = a; else r = b;  
    return r;  
}
```

Máximo

```
int max(int a, int b) {  
    if (a > b) return a;  
    return b;  
}
```

Blocos de comandos

- Em qualquer sítio onde é esperado um comando podemos colocar um *bloco de comandos*
- Um bloco de comandos é uma sequência de comandos ou declarações de variáveis entre chavetas
- As variáveis declaradas num bloco só podem ser usadas nesse bloco
- A definição de uma função é de facto um bloco de comandos

Máximo

```
int max(int a, int b) {  
    if (a > b) {  
        int r;  
        r = a;  
        return r;  
    } else {  
        int m;  
        m = b;  
        return m;  
    }  
}
```

Expressões lógicas

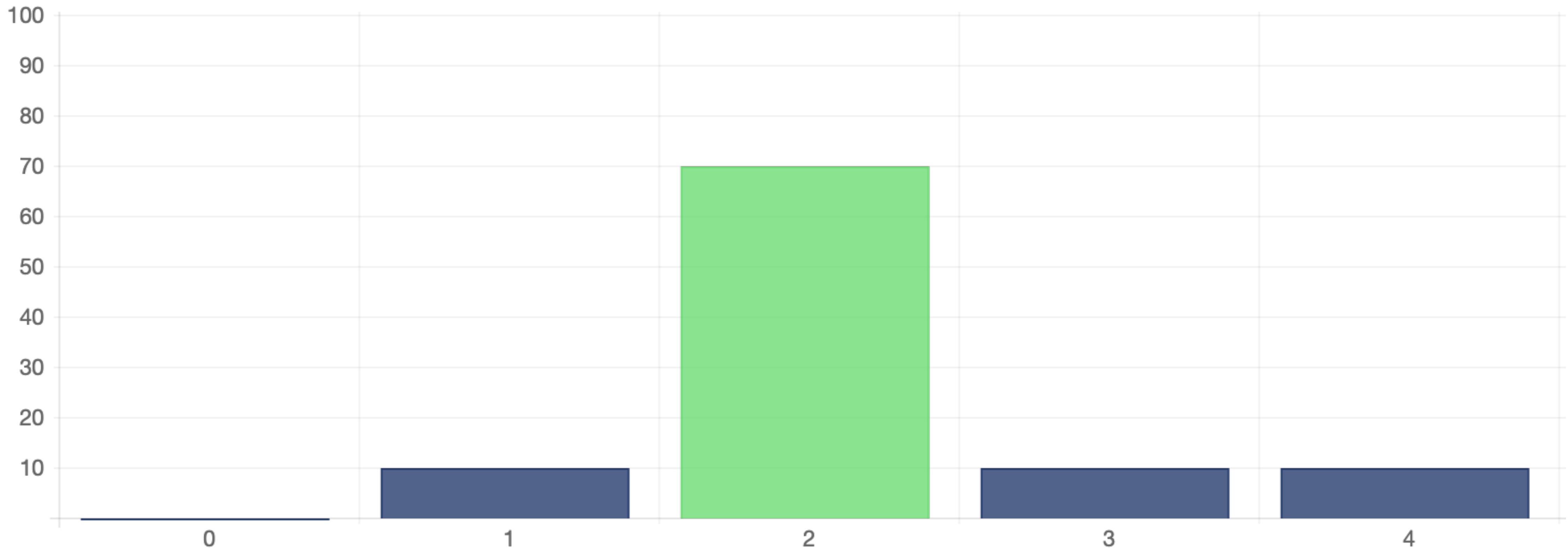
- A avaliação de uma expressão lógica é feita da esquerda para a direita
- Termina logo que seja possível determinar o valor da expressão

#4 Quantas comparações faz isalpha('0')?

```
int isalpha(int c) {  
    return ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'));  
}
```



#4 Quantas comparações faz `isalpha('0')`?



Ciclo while

```
while (expr) cmd;
```

Factorial

```
int fact(int n) {  
    int r = 1;  
    int i = 1;  
    while (i <= n) {  
        r = r * i;  
        i = i + 1;  
    }  
    return r;  
}
```

Factorial

```
int fact(int n) {  
    int r = 1;  
    int i = 1;  
    while (i <= n) {  
        r = r * i;  
        i = i + 1;  
    }  
    return r;  
}
```

Factorial

```
int fact(int n) {  
    int r = 1;  
    int i = 1;  
    while (i <= n) {  
        r = r * i;  
        i = i + 1;  
    }  
    return r;  
}
```

n	r	i	i <= n
---	---	---	--------

Factorial

```
int fact(int n) {  
    int r = 1;  
    int i = 1;  
    while (i <= n) {  
        r = r * i;  
        i = i + 1;  
    }  
    return r;  
}
```

n	r	i	i <= n
5	1	1	1

Factorial

```
int fact(int n) {  
    int r = 1;  
    int i = 1;  
    while (i <= n) {  
        r = r * i;  
        i = i + 1;  
    }  
    return r;  
}
```

n	r	i	i <= n
5	1	1	1
5	1	2	1

Factorial

```
int fact(int n) {  
    int r = 1;  
    int i = 1;  
    while (i <= n) {  
        r = r * i;  
        i = i + 1;  
    }  
    return r;  
}
```

n	r	i	i <= n
5	1	1	1
5	1	2	1
5	2	3	1

Factorial

```
int fact(int n) {  
    int r = 1;  
    int i = 1;  
    while (i <= n) {  
        r = r * i;  
        i = i + 1;  
    }  
    return r;  
}
```

n	r	i	i <= n
5	1	1	1
5	1	2	1
5	2	3	1
5	6	4	1

Factorial

```
int fact(int n) {  
    int r = 1;  
    int i = 1;  
    while (i <= n) {  
        r = r * i;  
        i = i + 1;  
    }  
    return r;  
}
```

n	r	i	i <= n
5	1	1	1
5	1	2	1
5	2	3	1
5	6	4	1
5	24	5	1

Factorial

```
int fact(int n) {  
    int r = 1;  
    int i = 1;  
    while (i <= n) {  
        r = r * i;  
        i = i + 1;  
    }  
    return r;  
}
```

n	r	i	i <= n
5	1	1	1
5	1	2	1
5	2	3	1
5	6	4	1
5	24	5	1
5	120	6	0

Factorial

```
int fact(int n) {  
    int r = 1;  
    while (n > 0) {  
        r = r * n;  
        n = n - 1;  
    }  
    return r;  
}
```

Factorial

```
int fact(int n) {  
    int r = 1;  
    while (n > 0) {  
        r = r * n;  
        n = n - 1;  
    }  
    return r;  
}
```

Factorial

```
int fact(int n) {  
    int r = 1;  
    while (n > 0) {  
        r = r * n;  
        n = n - 1;  
    }  
    return r;  
}
```

n	r	$n > 0$
---	---	---------

Factorial

```
int fact(int n) {  
    int r = 1;  
    while (n > 0) {  
        r = r * n;  
        n = n - 1;  
    }  
    return r;  
}
```

n	r	$n > 0$
5	1	1

Factorial

```
int fact(int n) {  
    int r = 1;  
    while (n > 0) {  
        r = r * n;  
        n = n - 1;  
    }  
    return r;  
}
```

n	r	$n > 0$
5	1	1
4	5	1

Factorial

```
int fact(int n) {  
    int r = 1;  
    while (n > 0) {  
        r = r * n;  
        n = n - 1;  
    }  
    return r;  
}
```

n	r	$n > 0$
5	1	1
4	5	1
3	20	1

Factorial

```
int fact(int n) {  
    int r = 1;  
    while (n > 0) {  
        r = r * n;  
        n = n - 1;  
    }  
    return r;  
}
```

n	r	n > 0
5	1	1
4	5	1
3	20	1
2	60	1

Factorial

```
int fact(int n) {  
    int r = 1;  
    while (n > 0) {  
        r = r * n;  
        n = n - 1;  
    }  
    return r;  
}
```

n	r	n > 0
5	1	1
4	5	1
3	20	1
2	60	1
1	120	1

Factorial

```
int fact(int n) {  
    int r = 1;  
    while (n > 0) {  
        r = r * n;  
        n = n - 1;  
    }  
    return r;  
}
```

n	r	n > 0
5	1	1
4	5	1
3	20	1
2	60	1
1	120	1
0	120	0

Operadores de atribuição

var = *var op expr*;

≡

var op= expr;

Factorial

```
int fact(int n) {  
    int r = 1;  
    int i = 1;  
    while (i <= n) {  
        r *= i;  
        i += 1;  
    }  
    return r;  
}
```

Comandos vs Expressões

- Em C qualquer expressão é também um comando
 - O comando pode não ter efeito no estado

```
int main() {  
    3+4;  
    return 0;  
}
```

- Uma atribuição é de facto uma expressão (e também um comando)
 - O = é um operador
 - O valor da expressão *var* = *expr* é o valor de *expr*
 - O efeito no estado é modificar o valor da variável *var*

Operadores ++ e --

Expressão	Valor	Efeito
<code>++x</code>	$x + 1$	$x = x + 1$
<code>x++</code>	x	$x = x + 1$
<code>--x</code>	$x - 1$	$x = x - 1$
<code>x--</code>	x	$x = x - 1$

Factorial

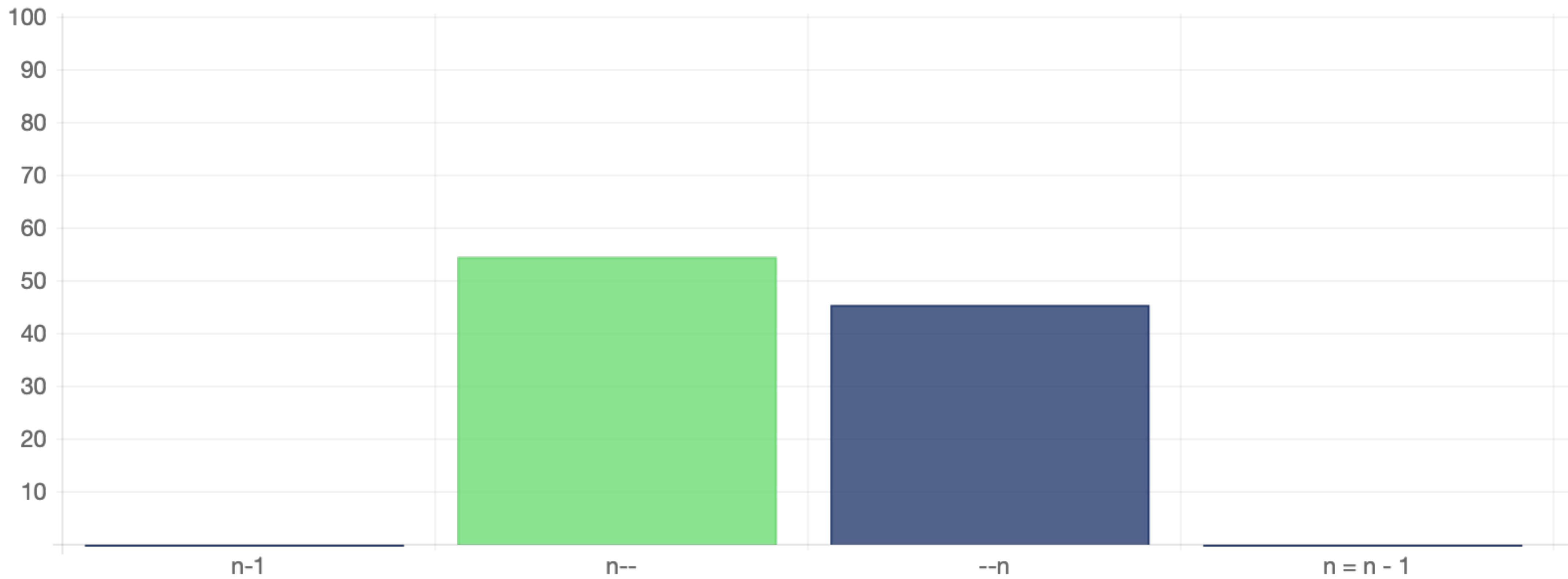
```
int fact(int n) {  
    int r, i;  
    r = i = 1;  
    while (i <= n) {  
        r *= i;  
        i++;  
    }  
    return r;  
}
```

#5 Que expressão usar para calcular o factorial?

```
int fact(int n) {  
    int r = 1;  
    while (n > 0) r *= [REDACTED];  
    return r;  
}
```

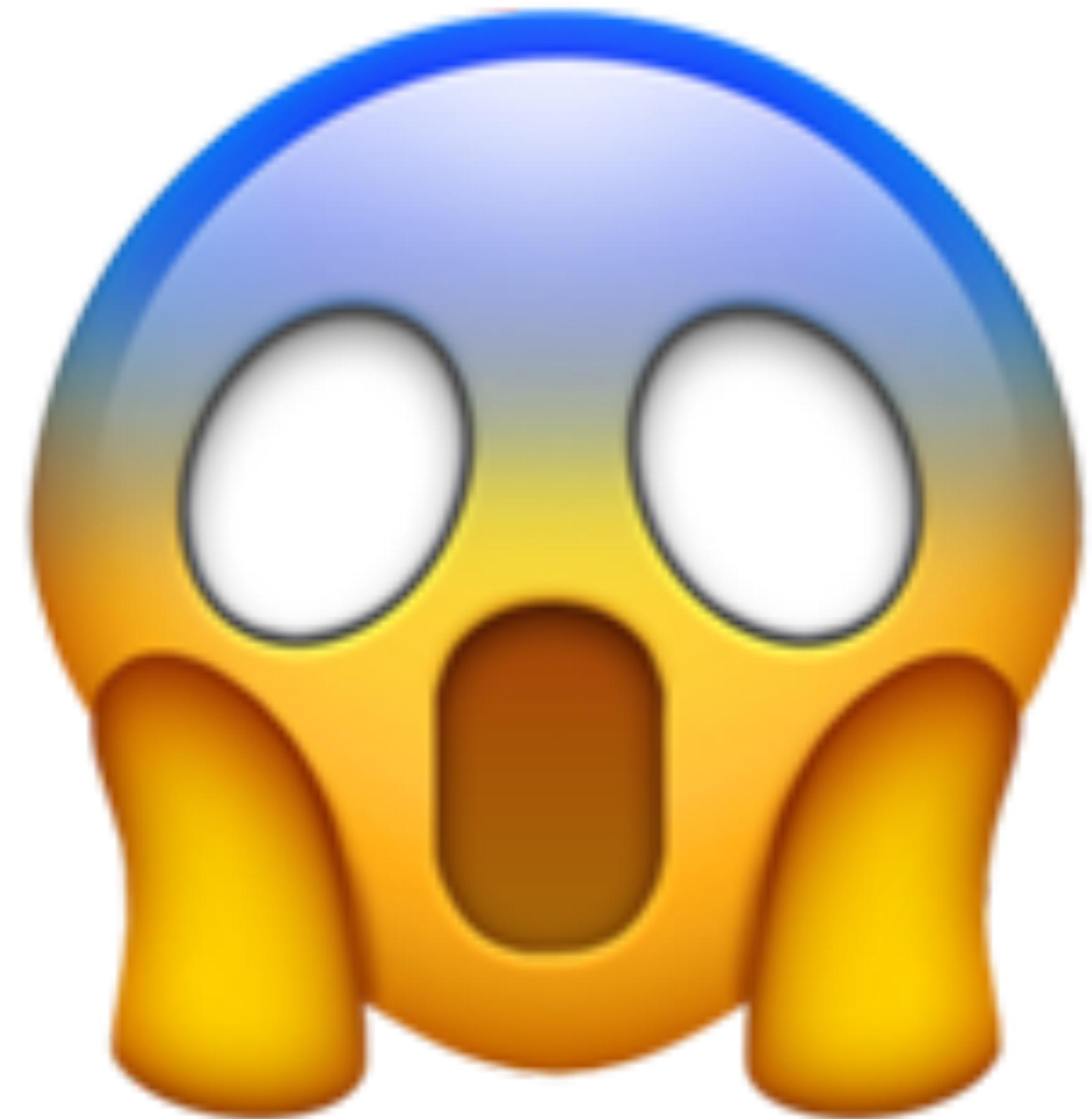


#5 Que expressão usar para calcular o factorial?



Factorial

```
int fact(int n) {  
    int r = 1;  
    while (n -= (r *= n) > 0);  
    return r;  
}
```



Aula 4

Ciclo for

init; while (cond) {cmd; iter;}

≡

for (*init; cond; iter*) *cmd*;

Factorial

```
int fact(int n) {  
    int r = 1;  
    int i = 1;  
    while (i <= n) {  
        r *= i;  
        i++;  
    }  
    return r;  
}
```

```
int fact(int n) {  
    int r = 1;  
    for (int i = 1; i <= n; i++)  
        r *= i;  
    return r;  
}
```

Factorial

```
int fact(int n) {  
    int r = 1;  
    while (n > 0) {  
        r *= n;  
        n--;  
    }  
    return r;  
}
```

```
int fact(int n) {  
    int r;  
    for (r = 1; n > 0; n--)  
        r *= n;  
    return r;  
}
```

Factorial

```
int fact(int n) {  
    int r;  
    for (r = 1; n > 0; r *= n, n--);  
    return r;  
}
```

Ciclo do-while

cmd; **while** (*cond*) *cmd;*

≡

do *cmd;* **while** (*cond*);

Factorial

```
int fact(int n) {  
    int r = 1, i = 0;  
    do {  
        i++;  
        r *= i;  
    }  
    while (i < n);  
    return r;  
}
```

break e **continue**

- O comando **break** termina a execução de um ciclo
 - A execução continua no primeiro comando depois do ciclo
- O comando **continue** termina a iteração actual
 - Num ciclo **while** ou **do-while** a execução continua no teste
 - Num ciclo **for** a execução continua no comando de incremento

Sorteio

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int r;
    while (1) {
        r = rand();                  // "sortear" um número entre 0 e RAND_MAX
        if (r == 0) break;
        if (r > 100) continue;
        printf("%d\n", r);
    }
    return 0;
}
```

Sorteio

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int r;
    while (1) {
        r = rand();                      // "sortear" um número entre 0 e RAND_MAX
        if (r == 0) break;
        if (r <= 100)
            printf("%d\n", r);
    }
    return 0;
}
```

Sorteio

```
#include <stdio.h>
#include <stdlib.h>

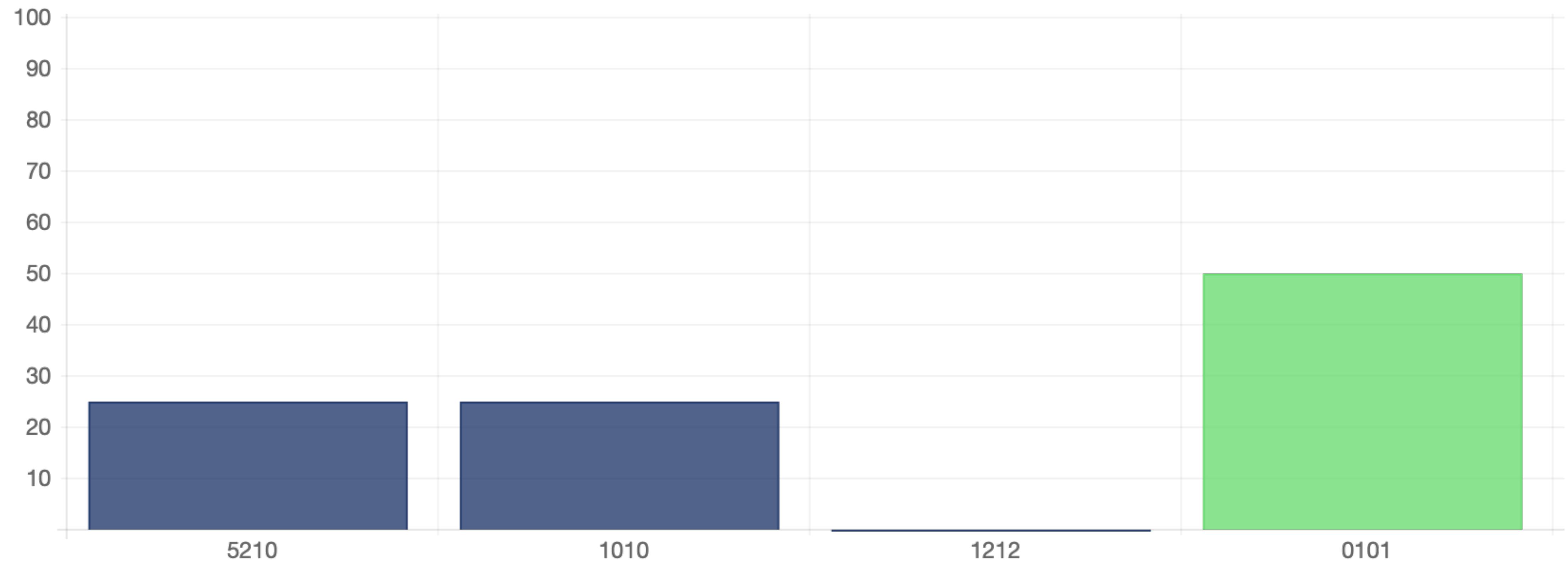
int main() {
    int r, ok = 1;
    while (ok) {
        r = rand();                      // "sortear" um número entre 0 e RAND_MAX
        if (r == 0) ok = 0;
        if (ok && r <= 100)
            printf("%d\n", r);
    }
    return 0;
}
```

#6 Que imprime proc(10)?

```
void proc(unsigned int n) {  
    for (; n > 0; n /= 2) printf("%d", n % 2);  
}
```



#6 Que imprime proc(10)?



Imprimir binário

```
int size(unsigned int n) {
    int size = 0;
    for (; n > 0; n /= 2) size++;
    return size;
}
int bit(unsigned int n, int i) {
    for (; i > 0; i--) n /= 2;
    return n % 2;
}
void binary(unsigned int n) {
    for (int i = size(n)-1; i >= 0; i--) {
        printf("%d", bit(n, i));
    }
}
```

Aula 5

Operadores lógicos *bitwise*

Operador	Significado	Exemplo (unsigned char)
<code>~</code>	Negação	<code>~10101001 == 01010110</code>
<code>&</code>	Conjunção	<code>10101001 & 11001010 == 10001000</code>
<code> </code>	Disjunção	<code>10101001 11001010 == 11101011</code>
<code>^</code>	Disjunção exclusiva	<code>10101001 ^ 11001010 == 01100011</code>
<code>>></code>	Shift para a direita	<code>10101001 >> 3 == 00010101</code>
<code><<</code>	Shift para a esquerda	<code>10101001 << 3 == 01001000</code>

Imprimir binário

```
int size(unsigned int n) {
    int size = 0;
    for (; n > 0; n >>= 1) size++;
    return size;
}
int bit(unsigned int n, int i) {
    return (n >> i) & 1;
}
void binary(unsigned int n) {
    for (int i = size(n)-1; i >= 0; i--) {
        printf("%d", bit(n, i));
    }
}
```

Recursividade

- Uma função pode invocar-se a si própria directa ou indirectamente



Factorial

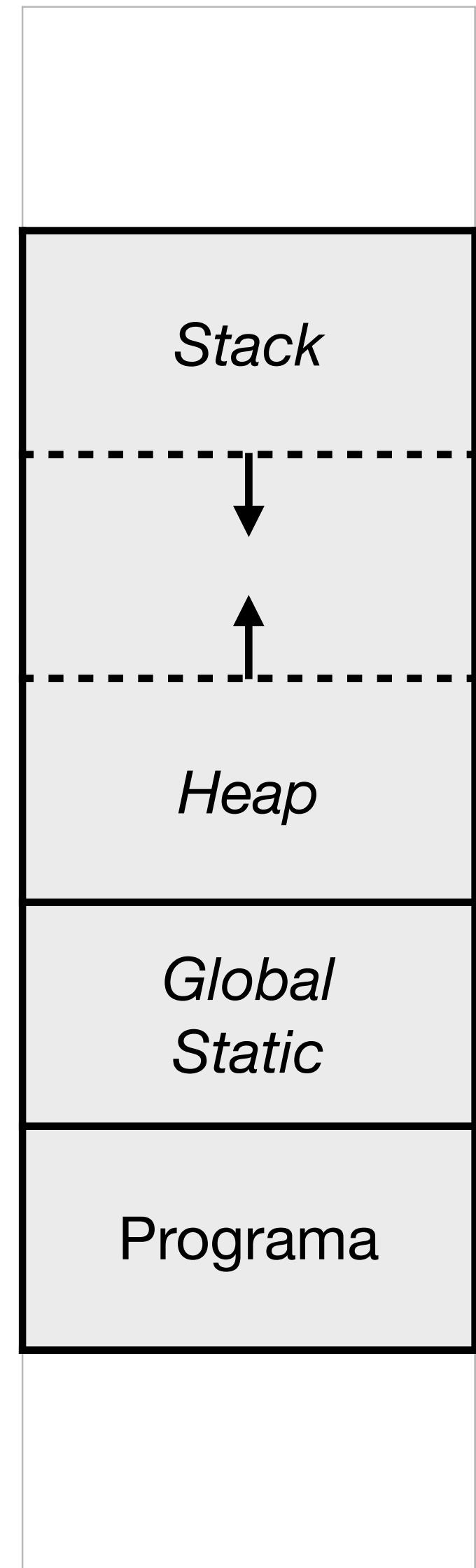
```
int fact(int n) {  
    int r;  
    if (n > 0) r = n * fact(n-1);  
    else r = 1;  
    return r;  
}
```

Imprimir binário

```
void binary(int n) {  
    if (n > 0) {  
        binary(n >> 1);  
        printf("%d", n & 1);  
    }  
}
```

Gestão de memória

- Automática
 - Memória reservada e libertada automaticamente na *stack* durante a execução
 - Variáveis locais e parâmetros
- Dinâmica
 - Memória reservada e libertada explicitamente na *heap* durante a execução
- Estática
 - Memória reservada em tempo de compilação e libertada apenas quando o programa termina
 - Variáveis globais e **static**



<https://pythontutor.com/c.html>



#7 Que imprime o seguinte programa?

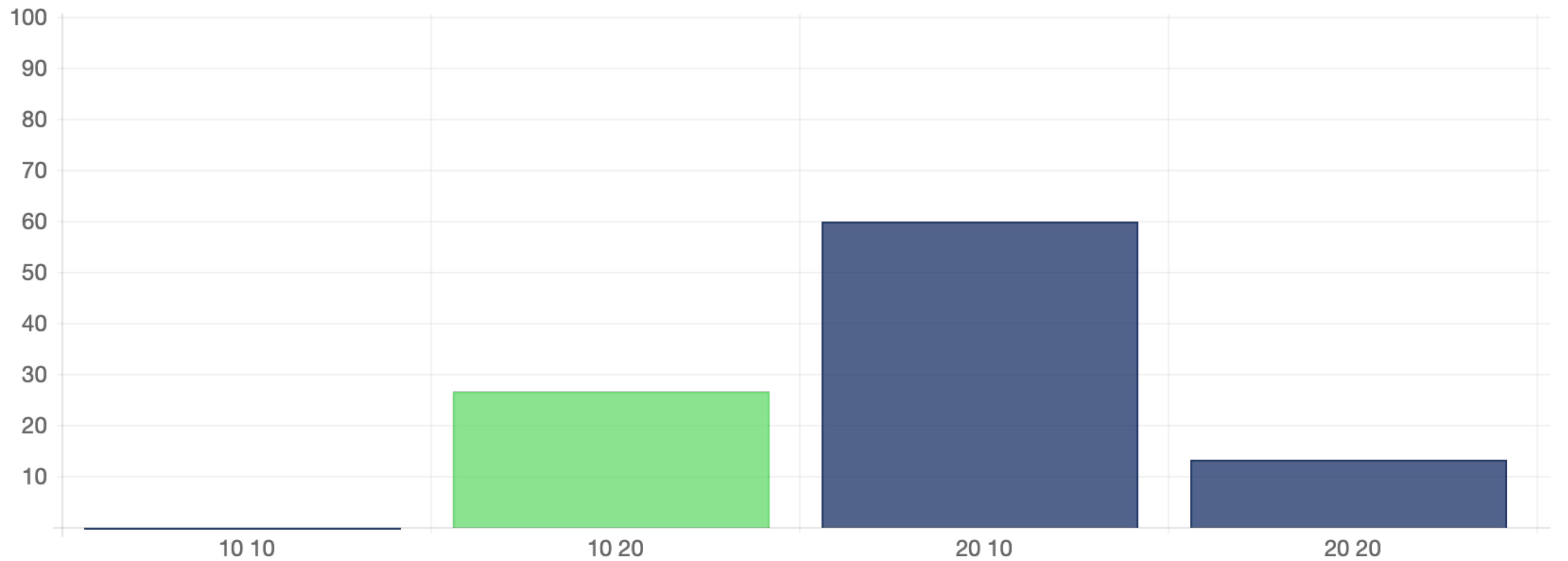
```
#include <stdio.h>

void swap(int x, int y) {
    int aux = y;
    y = x;
    x = aux;
}

int main() {
    int x = 10, y = 20;
    swap(x,y);
    printf("%d %d\n",x,y);
    return 0;
}
```



#7 Que imprime o seguinte programa?



Passagem de argumentos

- Passagem por valor (*call by value*)
 - O valor dos argumentos é copiado para os parâmetros da função
 - Uma modificação num parâmetro não afecta o argumento
- Passagem por referência (*call by reference*)
 - Uma referência (*alias*) para os argumentos é copiada para os parâmetros da função
 - Uma modificação num parâmetro afecta o argumento

Passagem de argumentos em C

- Em C temos sempre passagem por valor
- A passagem por referência pode ser simulada passando explicitamente (por valor) o endereço de memória do argumento
- Sabendo o endereço do argumento é possível na função alterar o seu conteúdo
- O endereço de uma variável é também conhecido como um *apontador* para a variável

Declaração de apontadores

*tipo *id;*

*tipo *id₀, *id₁, ...;*

void *id;

Operadores de referência

Operador	Significado
$*expr$	Conteúdo do endereço $expr$
$\&var$	Endereço de var

Swap

```
#include <stdio.h>

void swap(int *x, int *y) {
    int aux = *y;
    *y = *x;
    *x = aux;
}

int main() {
    int x = 10, y = 20;
    swap(&x,&y);
    printf("%d %d\n",x,y);
    return 0;
}
```

Swap

```
#include <stdio.h>

void swap(int *x, int *y) {
    int aux = *y;
    *y = *x;
    *x = aux;
}

int main() {
    int x = 10, y = 20;
    swap(&x,&y);
    printf("%d %d\n",x,y);
    return 0;
}
```

Endereço	Conteúdo	Variável	Função
16B8637E8	10	x	
16B8637E4	20	y	main

Swap

```
#include <stdio.h>

void swap(int *x, int *y) {
    int aux = *y;
    *y = *x;
    *x = aux;
}

int main() {
    int x = 10, y = 20;
    swap(&x,&y);
    printf("%d %d\n",x,y);
    return 0;
}
```

Endereço	Conteúdo	Variável	Função
16B8637E8	10	x	main
16B8637E4	20	y	
16B8637A4	16B8637E8	x	swap
16B8637A0	16B8637E4	y	
16B86379C	?	aux	

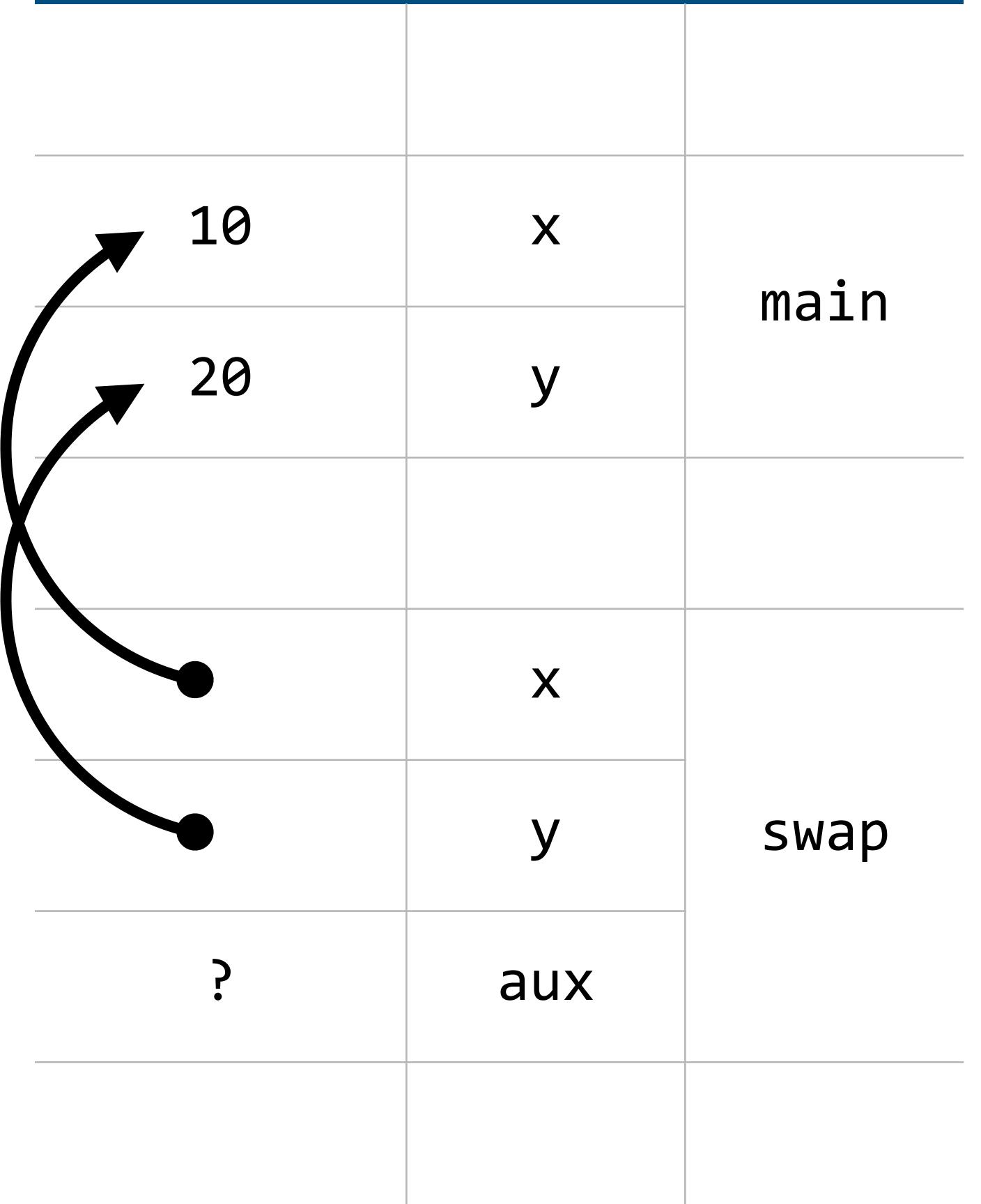
Swap

```
#include <stdio.h>

void swap(int *x, int *y) {
    int aux = *y;
    *y = *x;
    *x = aux;
}

int main() {
    int x = 10, y = 20;
    swap(&x,&y);
    printf("%d %d\n",x,y);
    return 0;
}
```

Conteúdo Variável Função



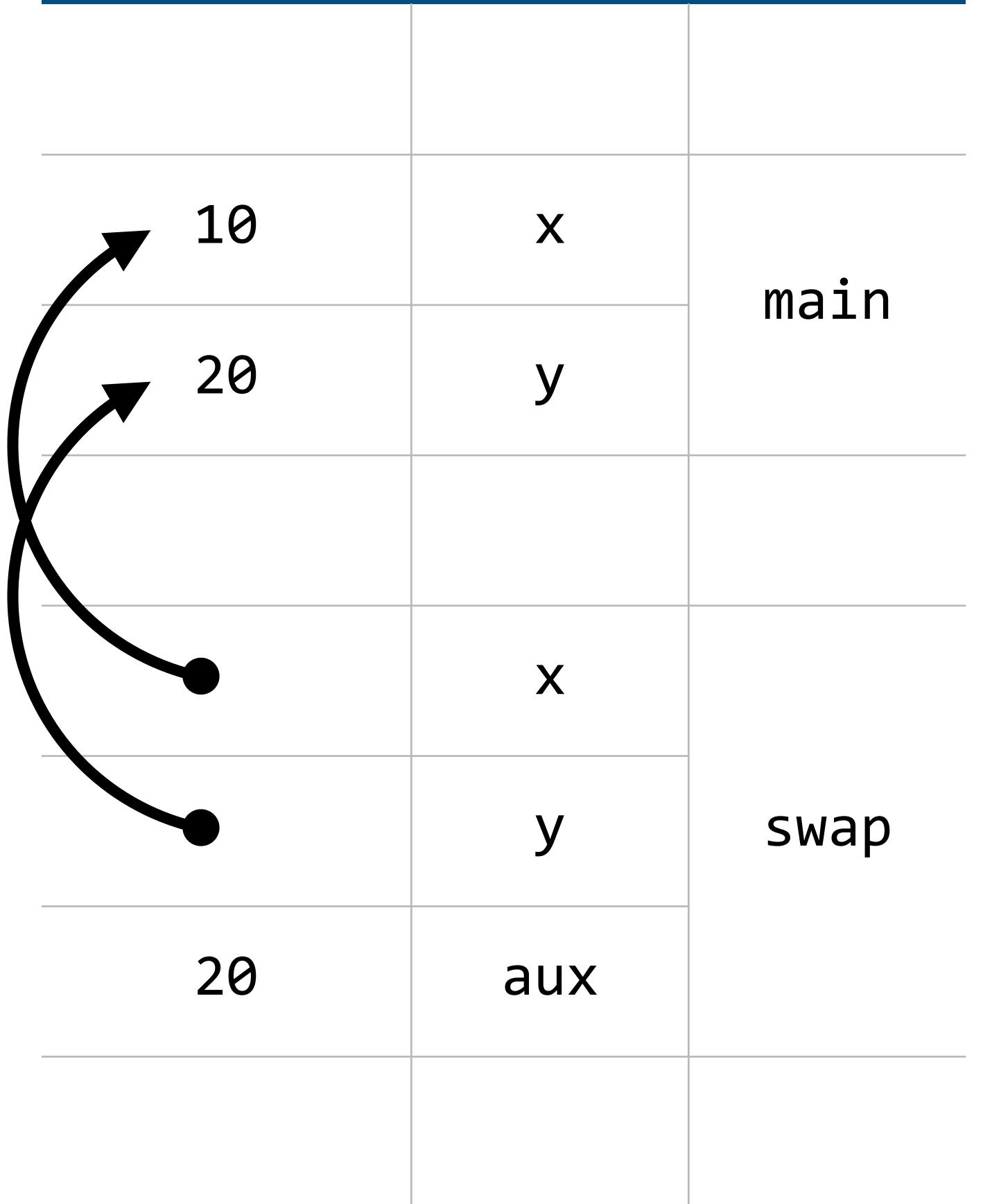
Swap

```
#include <stdio.h>

void swap(int *x, int *y) {
    int aux = *y;
    *y = *x;
    *x = aux;
}

int main() {
    int x = 10, y = 20;
    swap(&x,&y);
    printf("%d %d\n",x,y);
    return 0;
}
```

Conteúdo Variável Função



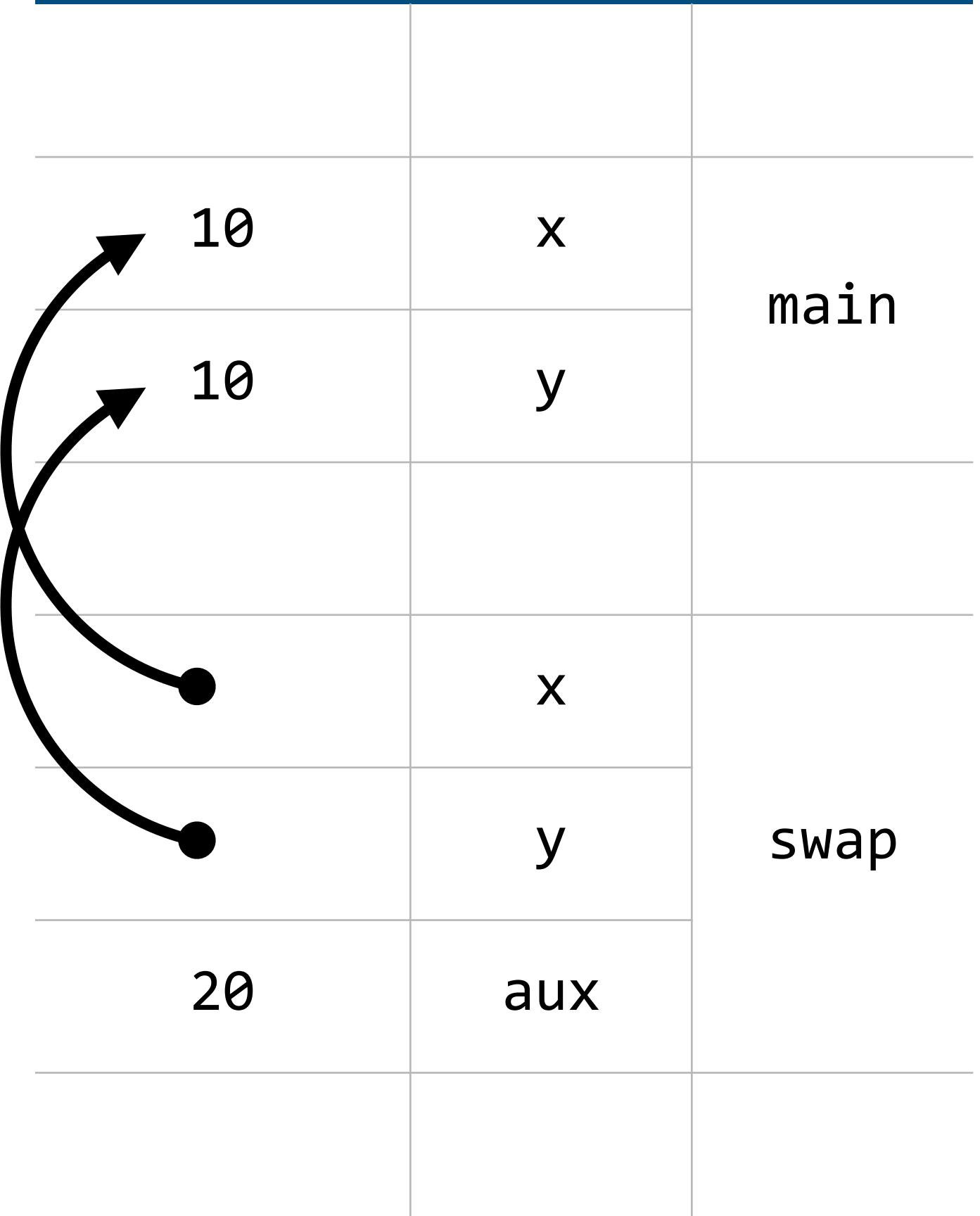
Swap

```
#include <stdio.h>

void swap(int *x, int *y) {
    int aux = *y;
    *y = *x;
    *x = aux;
}

int main() {
    int x = 10, y = 20;
    swap(&x,&y);
    printf("%d %d\n",x,y);
    return 0;
}
```

Conteúdo Variável Função



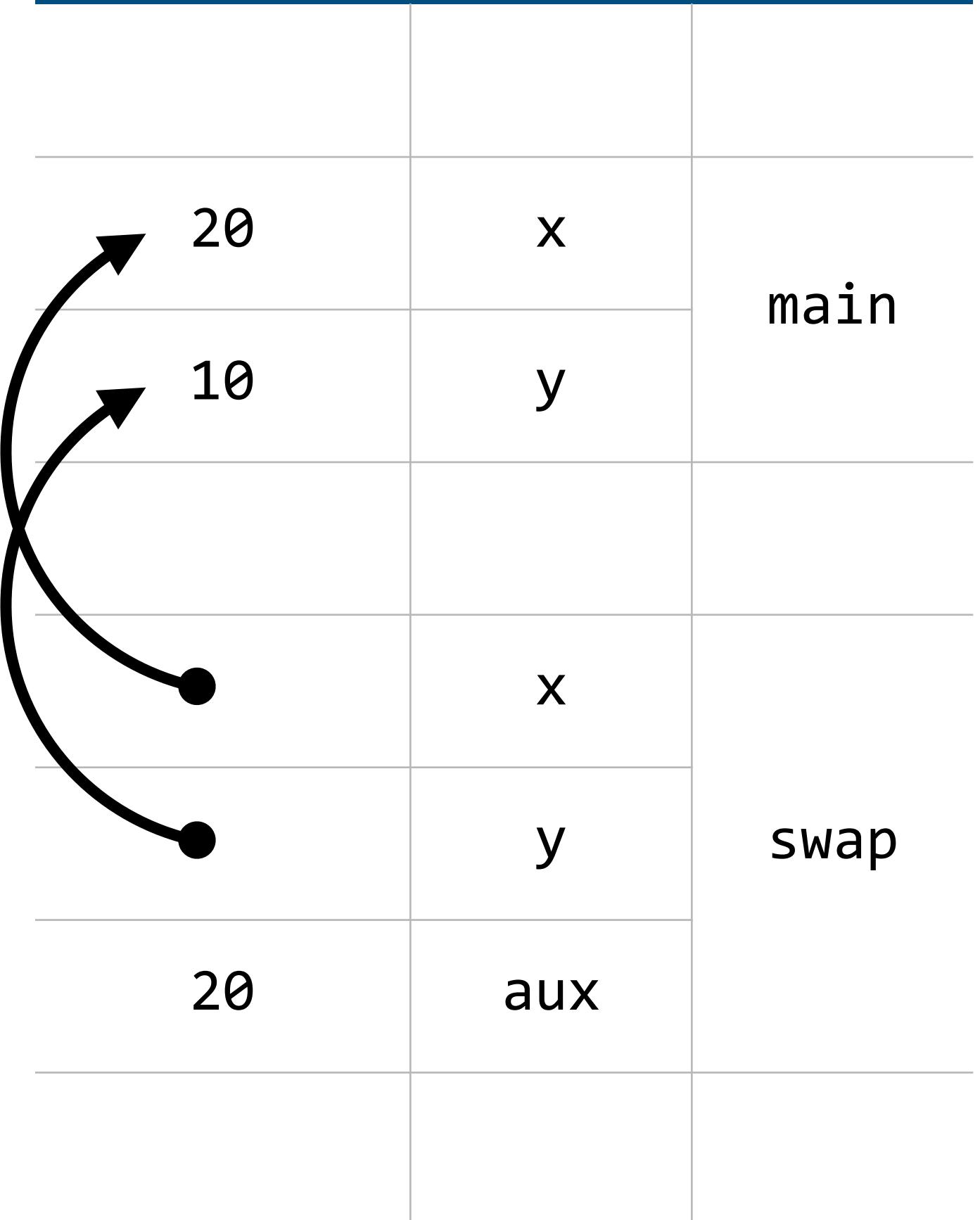
Swap

```
#include <stdio.h>

void swap(int *x, int *y) {
    int aux = *y;
    *y = *x;
    *x = aux;
}

int main() {
    int x = 10, y = 20;
    swap(&x,&y);
    printf("%d %d\n",x,y);
    return 0;
}
```

Conteúdo Variável Função



Swap

```
#include <stdio.h>

void swap(int *x, int *y) {
    int aux = *y;
    *y = *x;
    *x = aux;
}

int main() {
    int x = 10, y = 20;
    swap(&x,&y);
    printf("%d %d\n",x,y);
    return 0;
}
```

Conteúdo Variável Função

20	x	
10	y	main

A função scanf

```
#include <stdio.h>

int fact(int n) {
    int r = 1;
    for (; n > 0; n--) r *= n;
    return r;
}

int main() {
    int x;
    if (scanf("%d", &x) != 1) return 1;
    printf("fact(%d) == %d\n", x, fact(x));
    return 0;
}
```

Contar caracteres

```
#include <stdio.h>

int main() {
    int n = 0;
    char c;
    while ((c = getchar()) != '\n') n++;
    printf("%d\n", n);
    return 0;
}
```

#8 Como contar dígitos?

```
#include <stdio.h>

int main() {
    int n = 0;
    char c;
    while ((c = getchar()) != '\n') { [REDACTED] }
    printf("%d\n", n);
    return 0;
}
```



#8 Como contar dígitos?

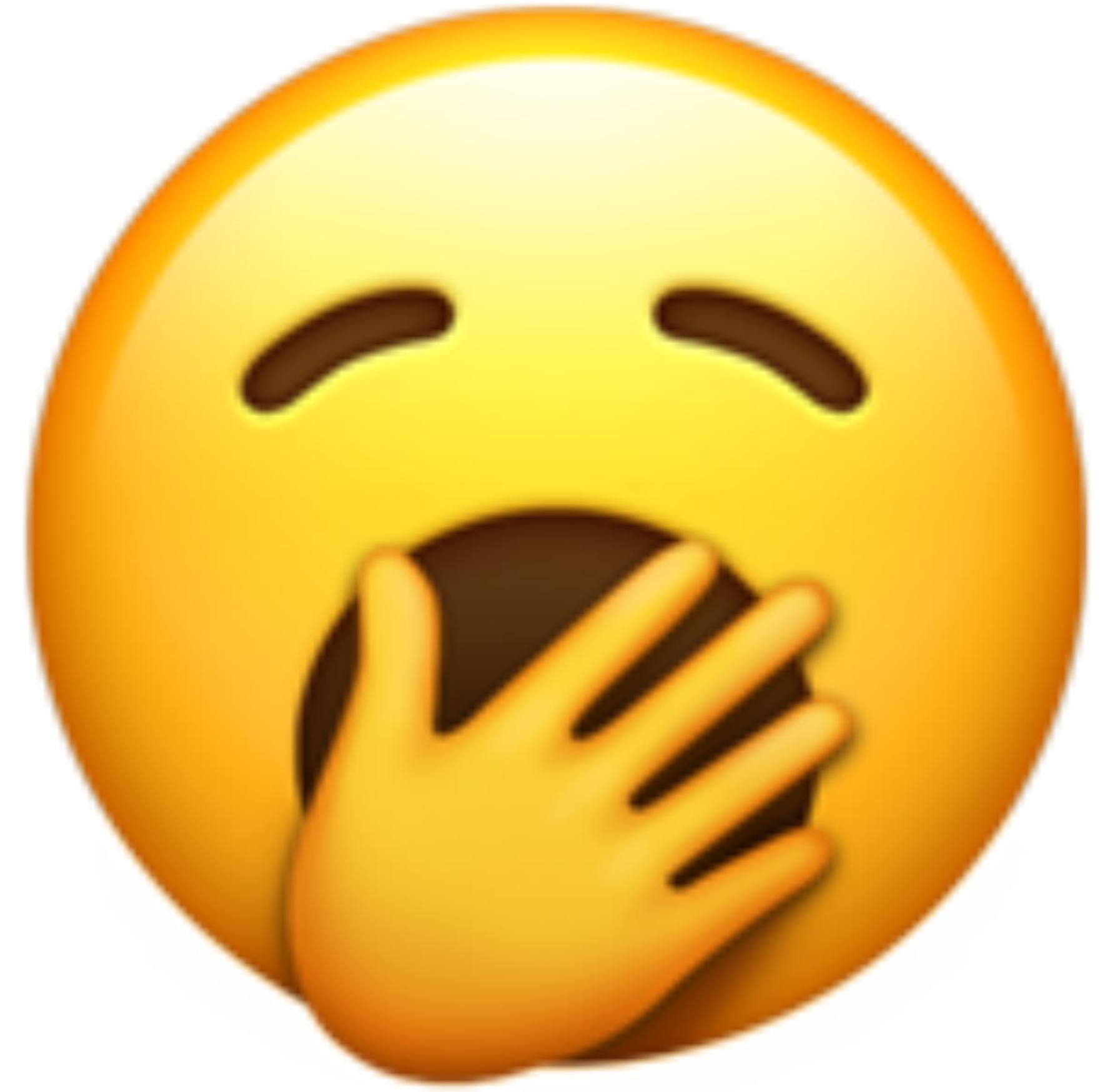


Aula 6

Contar dígitos separadamente

```
#include <stdio.h>

int main() {
    int n0 = 0, n1 = 0, ...;
    char c;
    while ((c = getchar()) != '\n') {
        n0 += (c == '0');
        n1 += (c == '1');
        ...
    }
    printf("0: %d\n", n0);
    printf("1: %d\n", n1);
    ...
    return 0;
}
```



Declaração de arrays

tipo id[n];

$id[0]$ $id[1]$ $id[2]$ $id[3]$... $id[n-1]$



Contar dígitos separadamente

```
#include <stdio.h>

int main() {
    int n[10], i;
    char c;
    for (i = 0; i < 10; i++) n[i] = 0;
    while ((c = getchar()) != '\n')
        if (c >= '0' && c <= '9') n[c-'0']++;
    for (i = 0; i < 10; i++) printf("%d: %d\n", i, n[i]);
    return 0;
}
```

Inicialização de arrays

tipo $id[n] = \{expr_0, expr_1, \dots, expr_k\};$

$id[0]$	$id[1]$...	$id[k]$...	$id[n-1]$
$expr_0$	$expr_1$		$expr_k$	0	0

Contar dígitos separadamente

```
#include <stdio.h>

int main() {
    int n[10] = {0}, i;
    char c;
    while ((c = getchar()) != '\n')
        if (c >= '0' && c <= '9') n[c-'0']++;
    for (i = 0; i < 10; i++) printf("%d: %d\n", i, n[i]);
    return 0;
}
```

Directiva #define

#define *id* *expr*

#define *id*(*id*₁, ..., *id*_{*n*}) *expr*

Sortear N números diferentes

```
#include <stdio.h>
#include <stdlib.h>

#define N 10

int existe(int x, int a[N]) {
    for (int i = 0; i < N; i++)
        if (a[i] == x) return 1;
    return 0;
}

int main() {
    int i = 0, x, p[N] = {0};
    while (i < N) {
        x = rand();
        if (x == 0) continue;
        if (existe(x,p)) continue;
        printf("%d\n", x);
        p[i++] = x;
    }
    return 0;
}
```

Aritmética de apontadores

- Quando se adiciona ou subtrai uma constante a um apontador, o tamanho da variável apontada é tido em consideração

`a = &x`

`a + n == (char *) a + n*sizeof(x)`

Arrays vs Apontadores

- O nome de um array corresponde ao endereço da primeira célula

$$a == \&(a[0])$$

- Quando é passado um array a uma função como argumento é de facto passado um apontador para a primeira célula
- O acesso a uma posição de um array é equivalente à deferência de um apontador

$$a[i] == *(a+i)$$

- Alterações ao array dentro da função afetam o array argumento
- O tamanho do array no parâmetro é irrelevante
- Se for necessário, o tamanho tem que ser passado num argumento

Pesquisa

```
int existe(int x, int a[], int n) {  
    for (int i = 0; i < n; i++)  
        if (a[i] == x) return 1;  
    return 0;  
}
```

Sortear N números diferentes

```
#include <stdio.h>
#include <stdlib.h>
#define N 10

int existe(int x, int a[], int n) {
    for (int i = 0; i < n; i++)
        if (a[i] == x) return 1;
    return 0;
}
```

```
int main() {
    int i = 0, x, p[N];
    while (i < N) {
        x = rand();
        if (x == 0) continue;
        if (existe(x,p,i)) continue;
        printf("%d\n", x);
        p[i++] = x;
    }
    return 0;
}
```

Pesquisa

```
int existe(int x, int *a, int n) {  
    for (int i = 0; i < n; i++)  
        if (*(a+i) == x) return 1;  
    return 0;  
}
```

Pesquisa

```
int existe(int x, int *a, int n) {  
    for (int i = 0; i < n; i++, a++)  
        if (*a == x) return 1;  
    return 0;  
}
```

Pesquisa

```
int existe(int x, int *a, int n) {  
    if (n == 0) return 0;  
    if (*a == x) return 1;  
    return existe(x, a+1, n-1);  
}
```

<https://pythontutor.com/c.html>



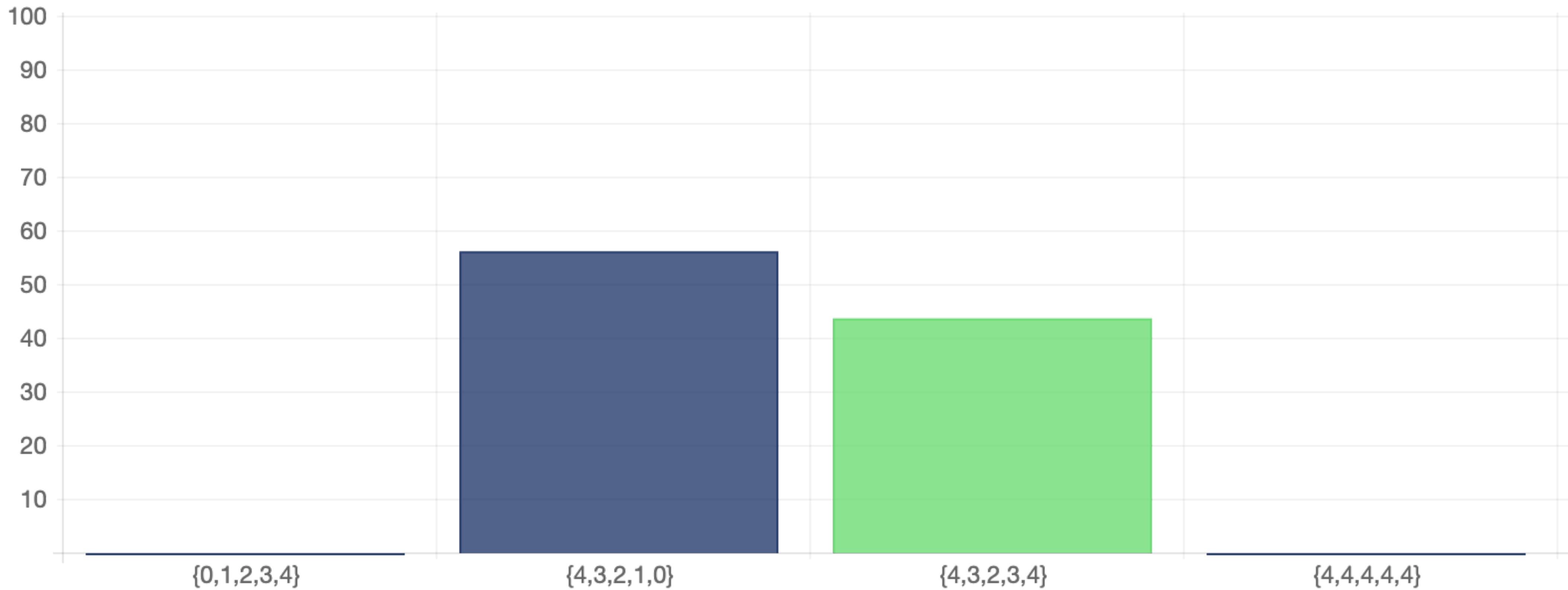
#9 Qual o conteúdo de a depois do modifica(a,5)?

```
void modifica(int b[], int n) {  
    int i = 0, j = n-1;  
    while (i < n) {  
        b[i] = b[j];  
        i++; j--;  
    }  
}
```

```
int main() {  
    int a[5] = {0,1,2,3,4};  
    modifica(a,5);  
    return 0;  
}
```



#9 Qual o conteúdo de a depois do modifica(a,5)?



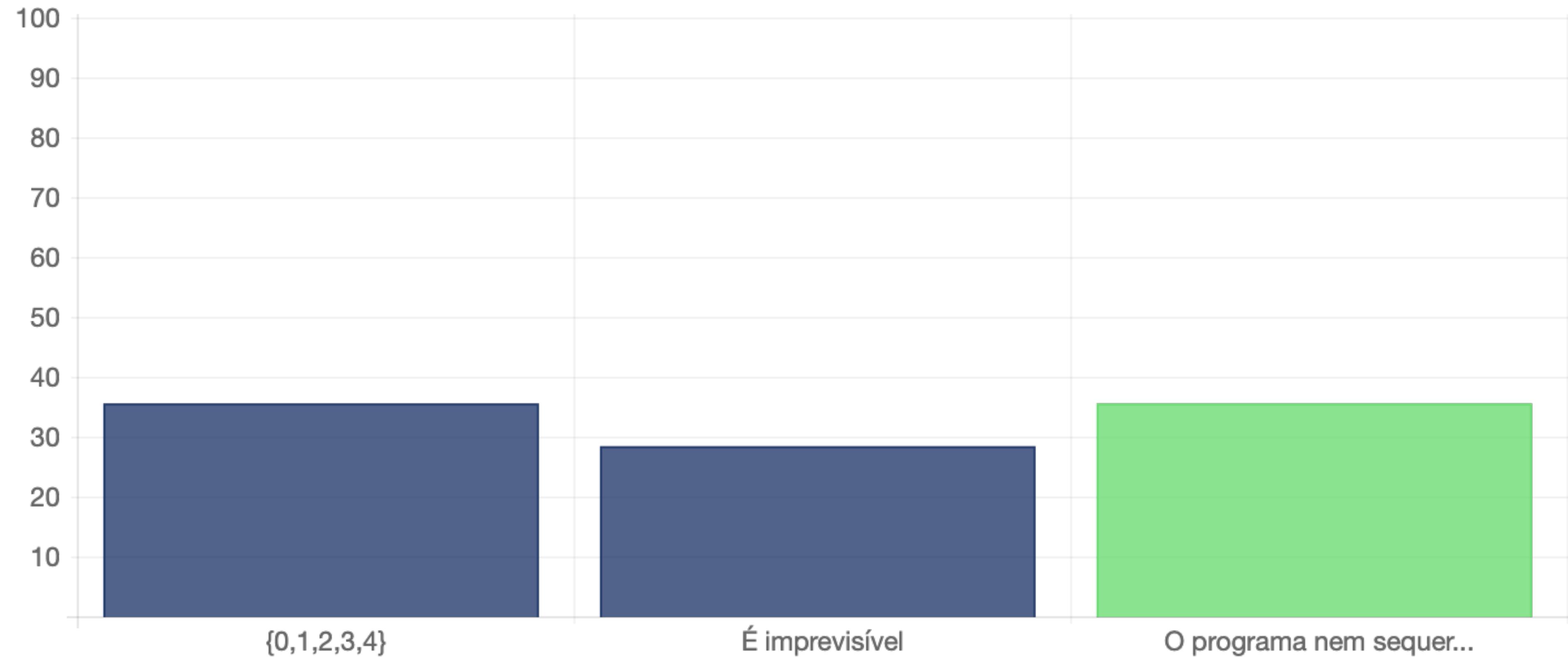
#10 Qual o conteúdo de b depois do copia(a,5)?

```
int *copia(int a[], int n) {  
    int b[n];  
    for (int i = 0; i < n; i++) b[i] = a[i];  
    return b;  
}
```

```
int main() {  
    int a[5] = {0,1,2,3,4};  
    int b[5];  
    b = copia(a,5);  
    return 0;  
}
```



#10 Qual o conteúdo de b depois do copia(a,5)?



Arrays vs Apontadores

- Na declaração de um parâmetro de uma função, `int a[N]` é a mesma coisa que `int *a`
 - Na stack é alocada 1 célula de memória para a com capacidade para um endereço
 - Dado um $0 \leq i < N$, podemos modificar `a[i]` (modifica a i -ésima célula do array argumento)
 - Podemos também modificar a (modifica a cópia local do endereço do array argumento)

Arrays vs Apontadores

- Mas na declaração de uma variável não é a mesma coisa!
- A declaração `int a[N]` aloca N variáveis do tipo `int`
 - Embora `a == &(a[0])`, não é alocada memória para a
 - Dado um $0 \leq i < N$, podemos modificar `a[i]`,
 - Mas não é possível modificar a
- A declaração `int *a` aloca 1 variável com capacidade para um endereço
 - Podemos modificar a
 - Mas não faz sentido modificar `a[i]`, a não ser que a aponte para um array

Copiar um array

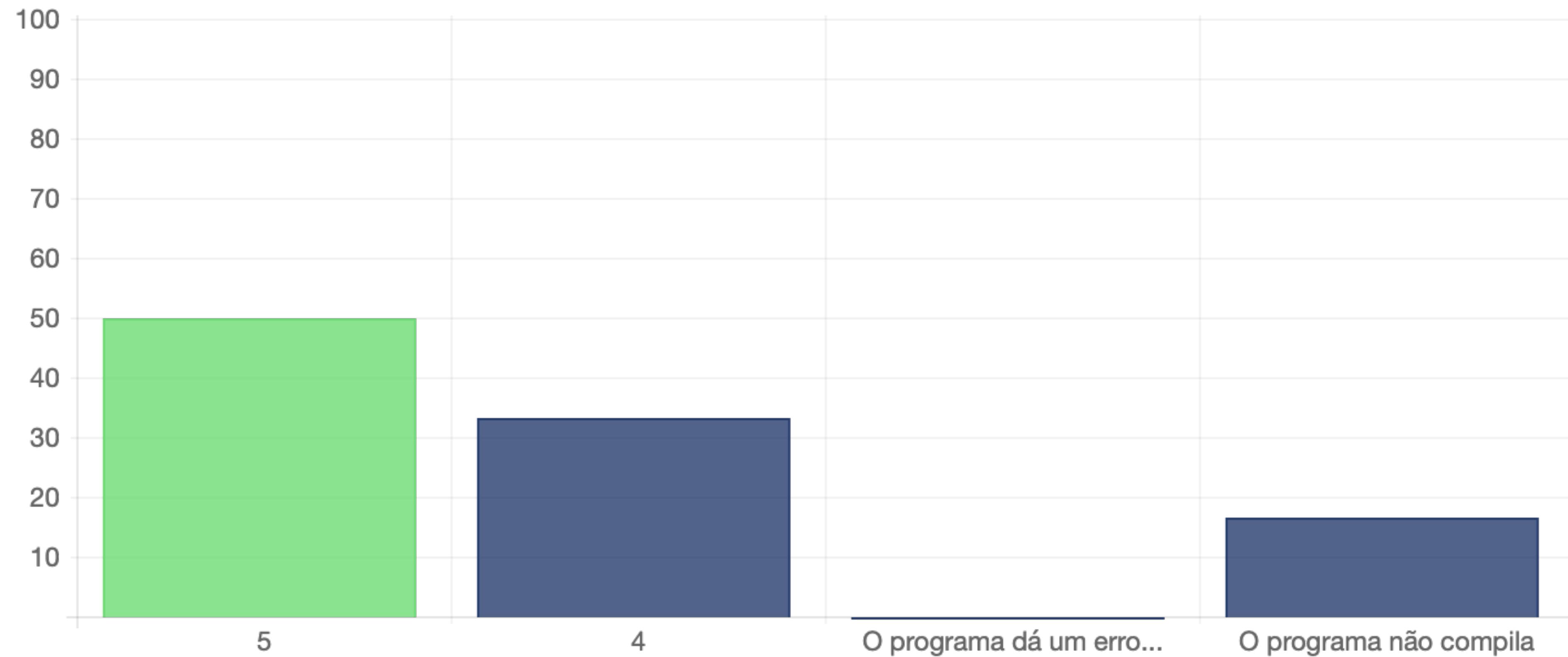
```
void copia(int a[], int b[], int n) {  
    for (int i = 0; i < n; i++) b[i] = a[i];  
}  
  
int main() {  
    int a[5] = {0,1,2,3,4};  
    int b[5];  
    copia(a,b,5);  
    return 0;  
}
```

#11 Qual o conteúdo de a[4] no final?

```
int main() {  
    int a[5] = {0,1,2,3,4};  
    int *b = a;  
    b[4] = 5;  
    return 0;  
}
```



#11 Qual o conteúdo de `a[4]` no final?



Aula 7

Strings

- Uma string é um array de caracteres
- Para ser bem formada deve terminar com o caracter '\0'

```
char s[6] = "ola";
```

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]
'o'	'l'	'a'	'\0'		

strlen

```
int strlen(char s[]) {  
    int i;  
    for (i = 0; s[i] != '\0'; i++);  
    return i;  
}
```

strlen

```
int strlen(char *s) {  
    char *t = s;  
    for (; *t; t++);  
    return t-s;  
}
```

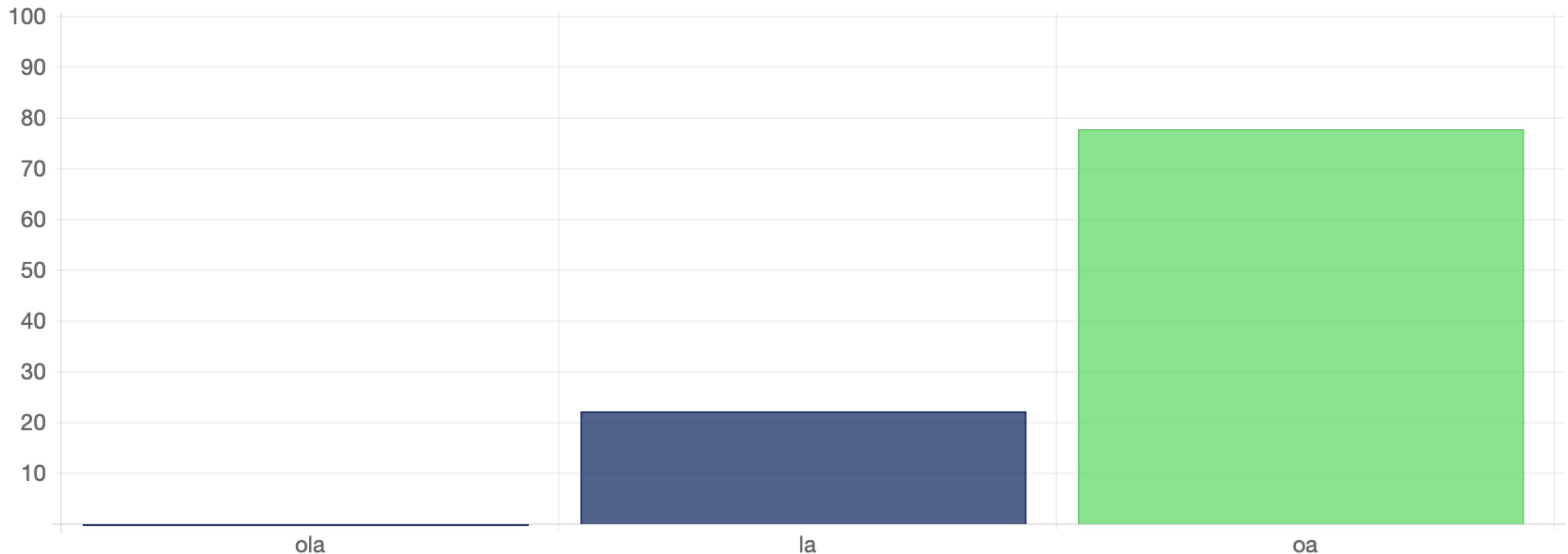
#12 Que imprime o seguinte programa?

```
void func(char s[], int i) {
    for (; s[i] != '\0'; i++) s[i] = s[i+1];
}

int main() {
    char s[10] = "ola";
    func(s,1);
    printf("%s\n", s);
    return 0;
}
```



#12 Que imprime o seguinte programa?



Remover i-ésimo caracter

```
void delete(char s[], int i) {  
    for (; s[i] != '\0'; i++) s[i] = s[i+1];  
}
```

```
int main() {  
    char s[10] = "ola";  
    delete(s,1);  
    printf("%s\n", s);  
    return 0;  
}
```

Remover i-ésimo caracter

```
char *delete(char s[], int i) {
    for (; s[i] != '\0'; i++) s[i] = s[i+1];
    return s;
}

int main() {
    char s[10] = "ola";
    printf("%s\n", delete(s,1));
    return 0;
}
```

Remover espaços

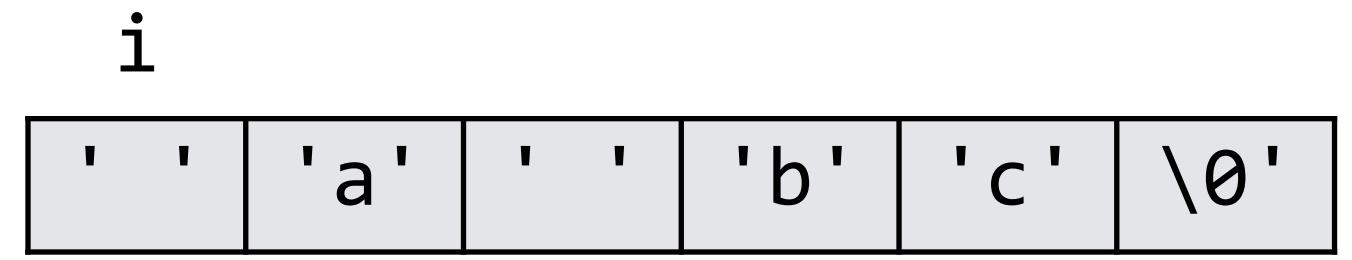
```
#include <ctype.h>

char *trim(char s[]) {
    int i = 0;
    while (s[i] != '\0') {
        if (isspace(s[i])) delete(s,i);
        else i++;
    }
    return s;
}
```

Remover espaços

```
#include <ctype.h>

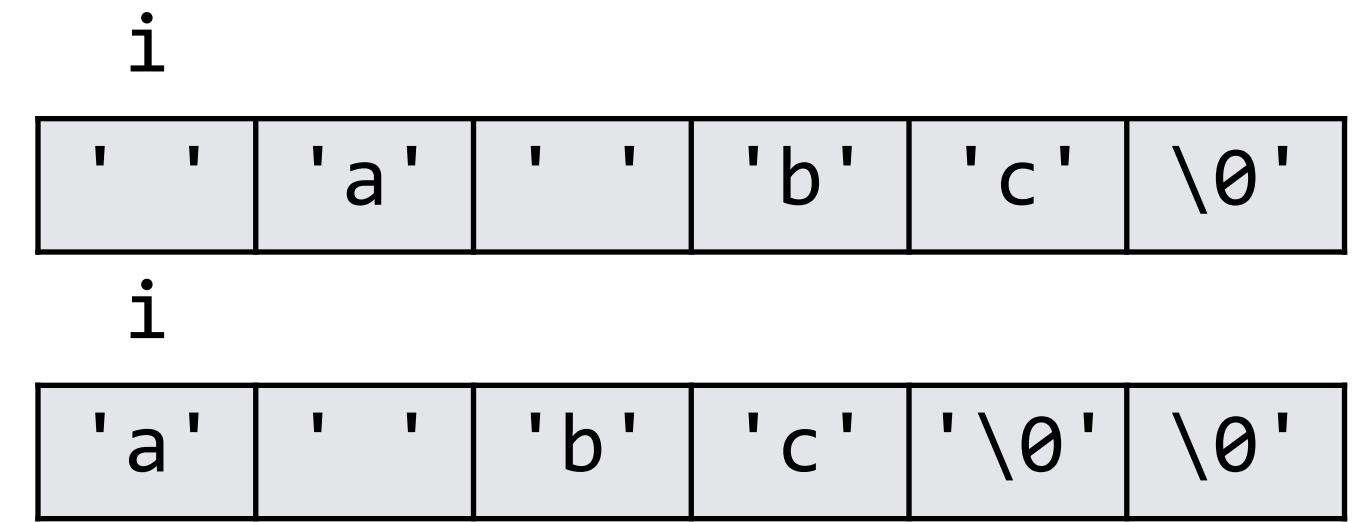
char *trim(char s[]) {
    int i = 0;
    while (s[i] != '\0') {
        if (isspace(s[i])) delete(s,i);
        else i++;
    }
    return s;
}
```



Remover espaços

```
#include <ctype.h>

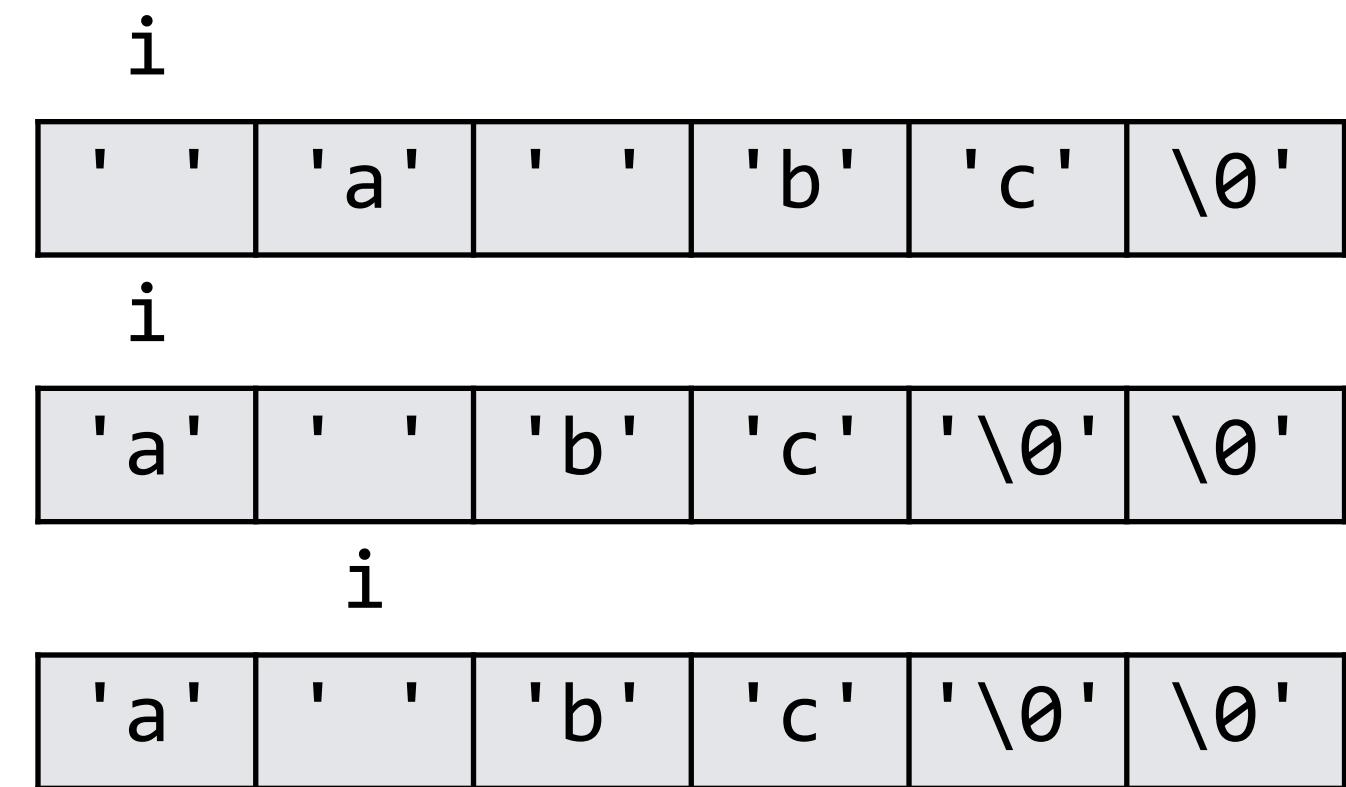
char *trim(char s[]) {
    int i = 0;
    while (s[i] != '\0') {
        if (isspace(s[i])) delete(s,i);
        else i++;
    }
    return s;
}
```



Remover espaços

```
#include <ctype.h>

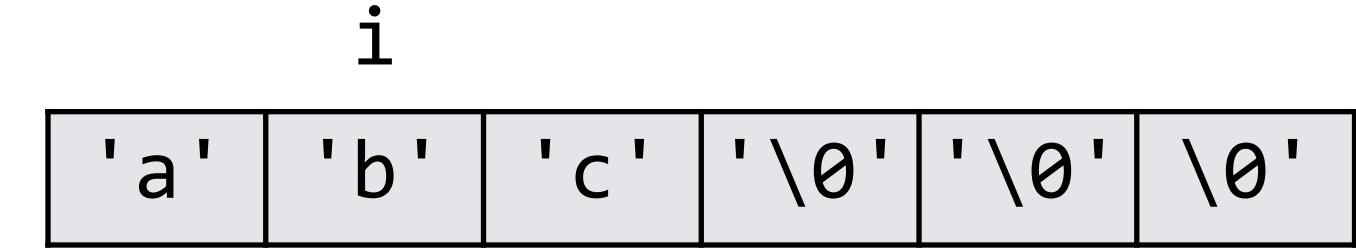
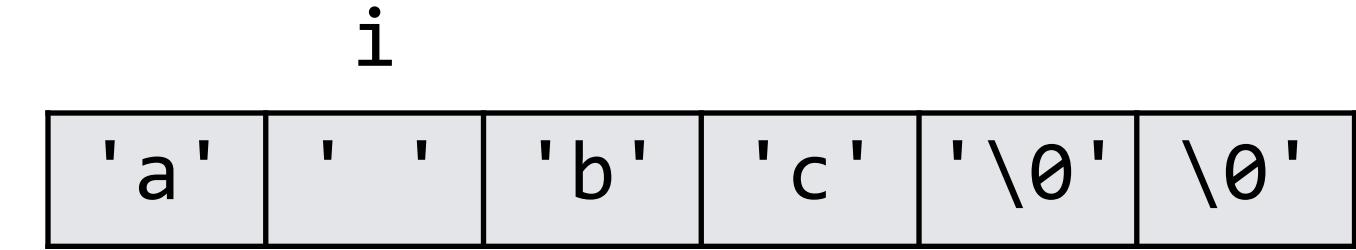
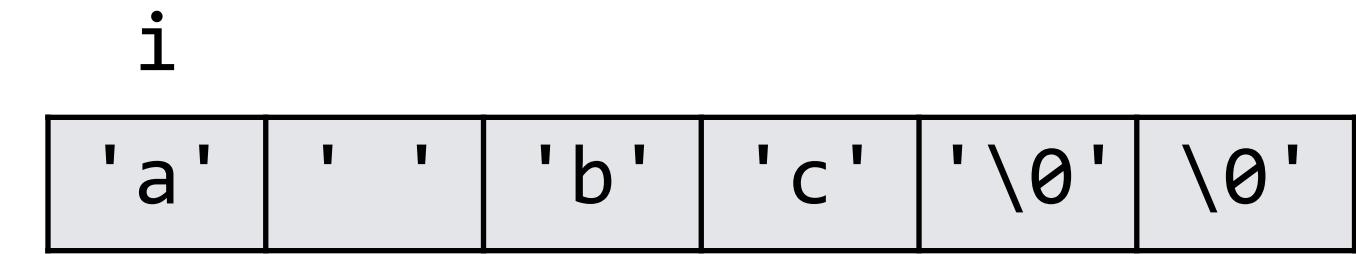
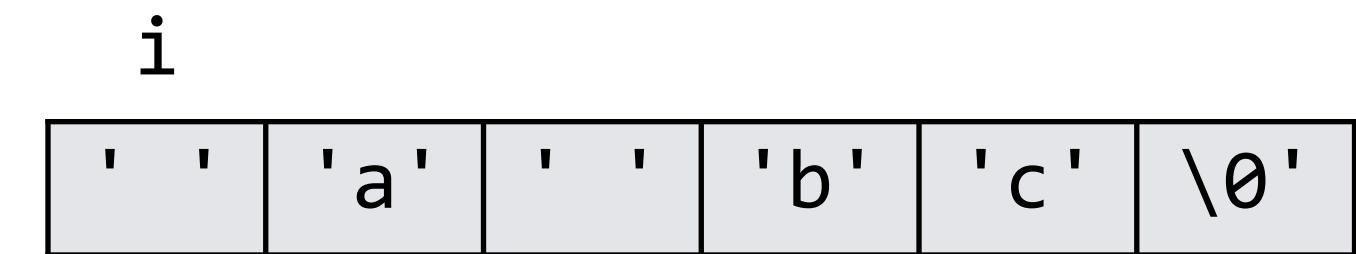
char *trim(char s[]) {
    int i = 0;
    while (s[i] != '\0') {
        if (isspace(s[i])) delete(s,i);
        else i++;
    }
    return s;
}
```



Remover espaços

```
#include <ctype.h>

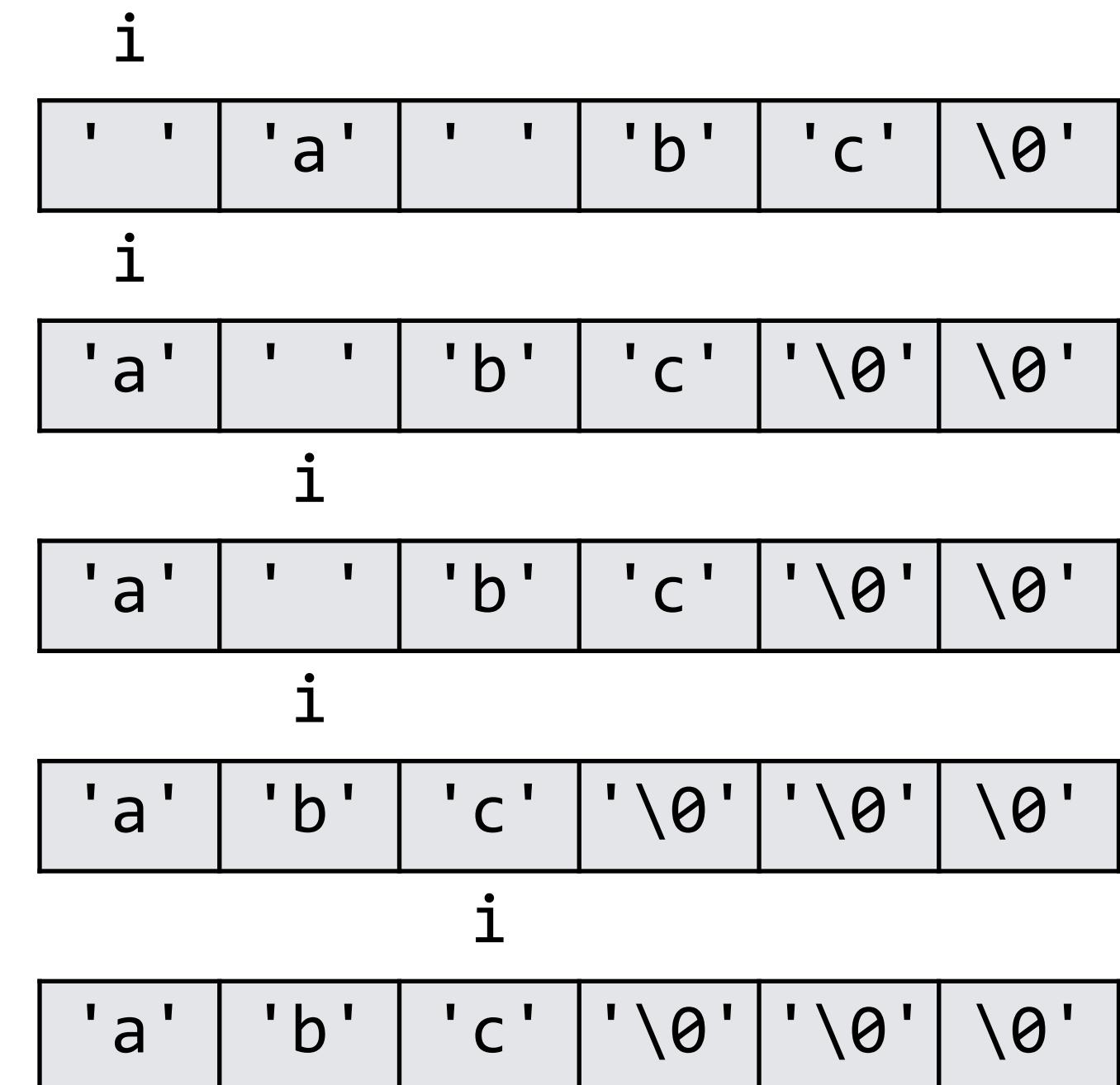
char *trim(char s[]) {
    int i = 0;
    while (s[i] != '\0') {
        if (isspace(s[i])) delete(s,i);
        else i++;
    }
    return s;
}
```



Remover espaços

```
#include <ctype.h>

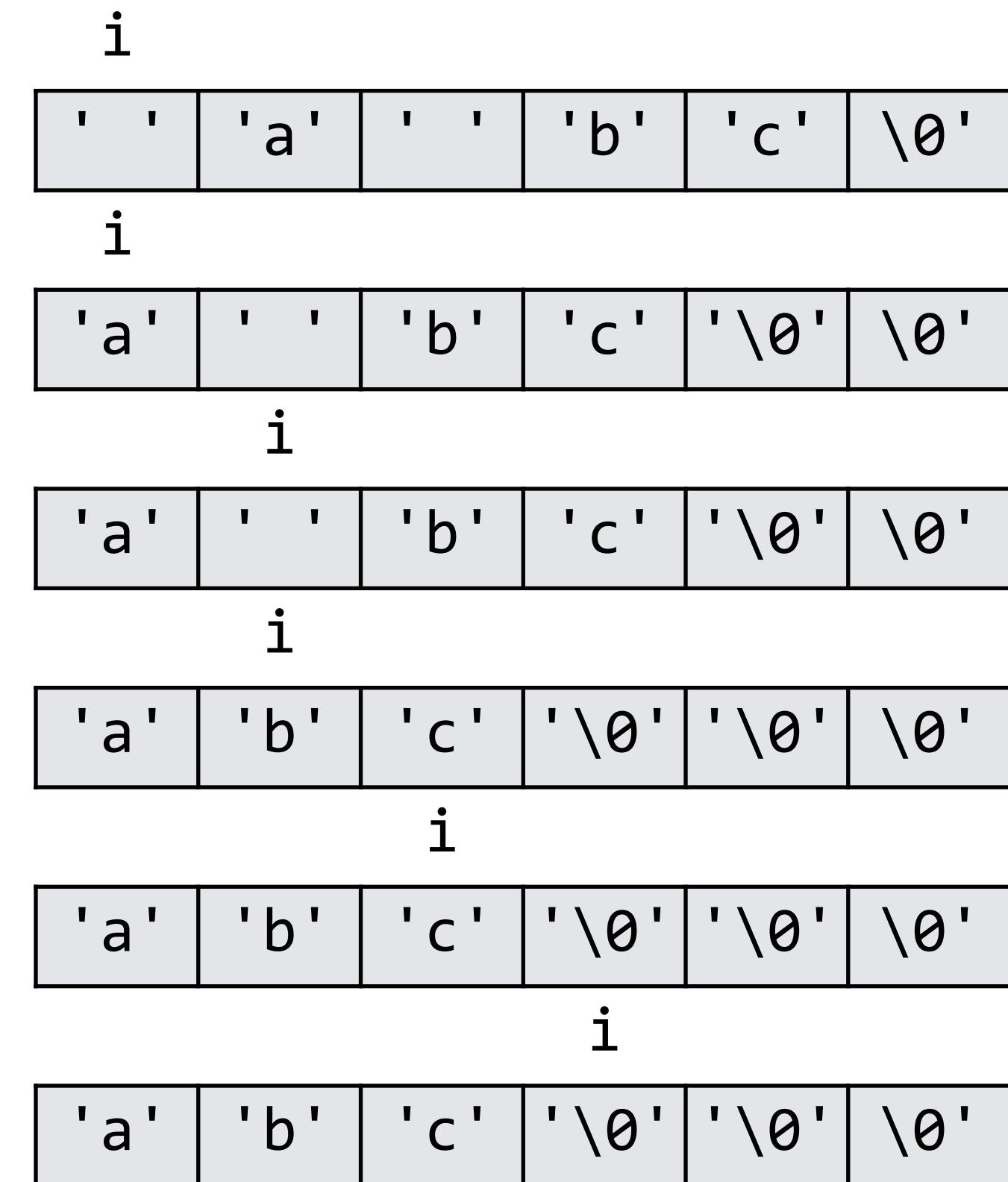
char *trim(char s[]) {
    int i = 0;
    while (s[i] != '\0') {
        if (isspace(s[i])) delete(s,i);
        else i++;
    }
    return s;
}
```



Remover espaços

```
#include <ctype.h>

char *trim(char s[]) {
    int i = 0;
    while (s[i] != '\0') {
        if (isspace(s[i])) delete(s,i);
        else i++;
    }
    return s;
}
```



Remover espaços

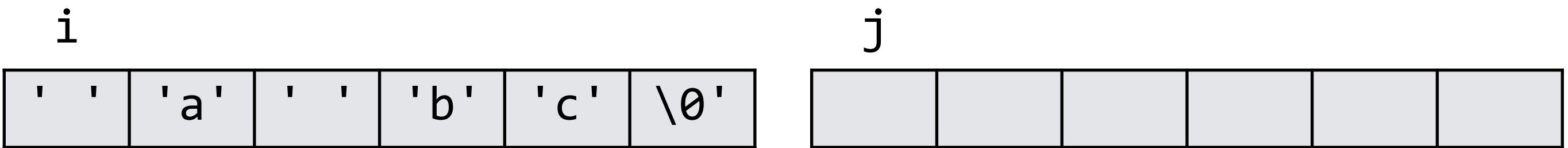
```
#include <ctype.h>

char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```

Remover espaços

```
#include <ctype.h>

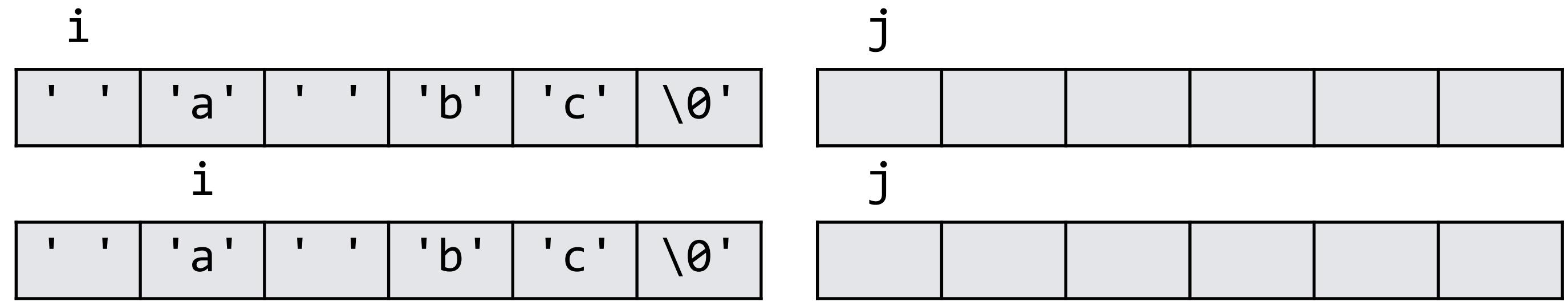
char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

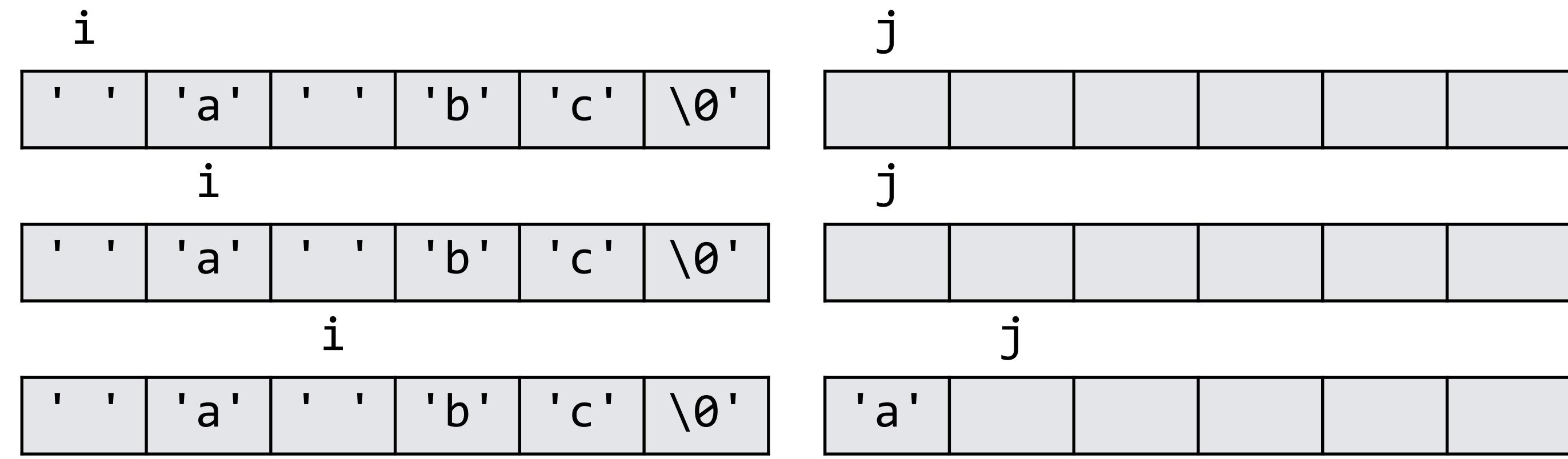
char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

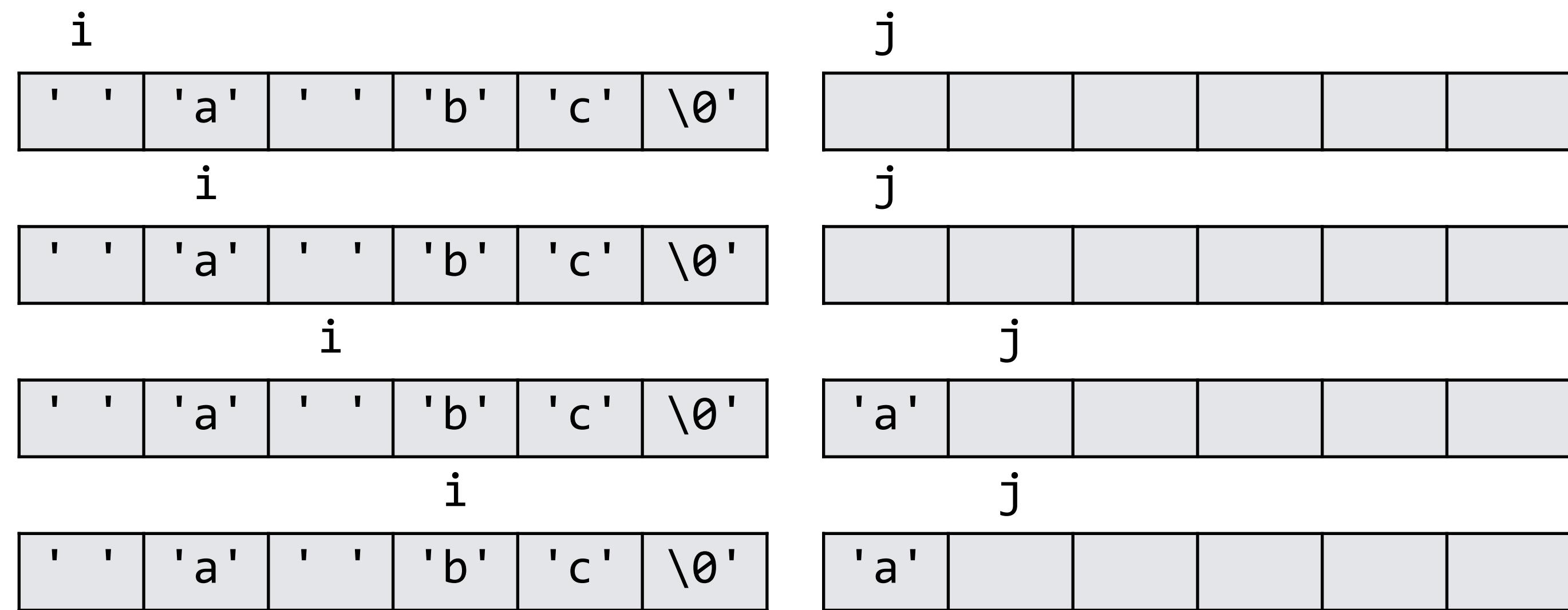
char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

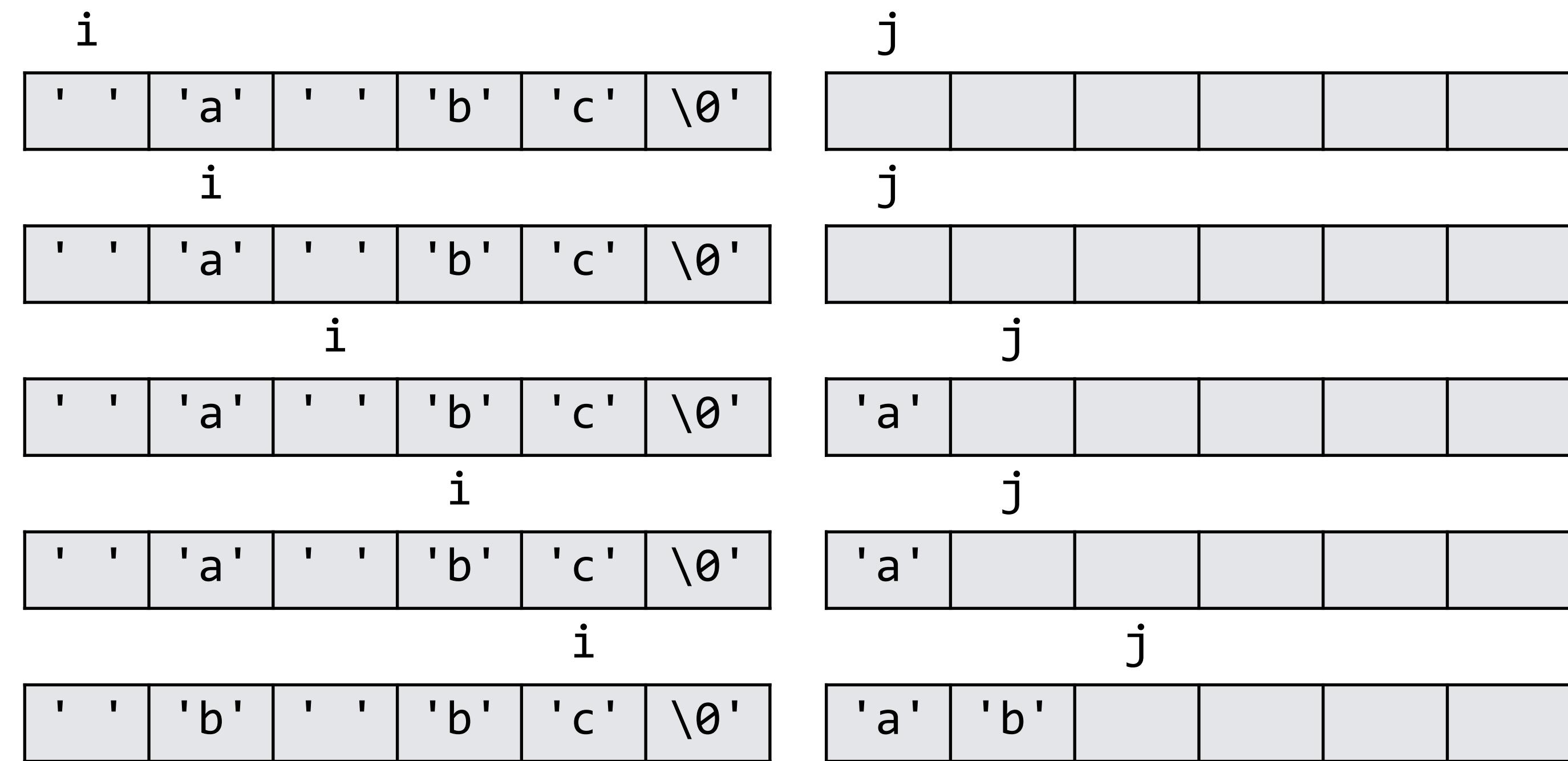
char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

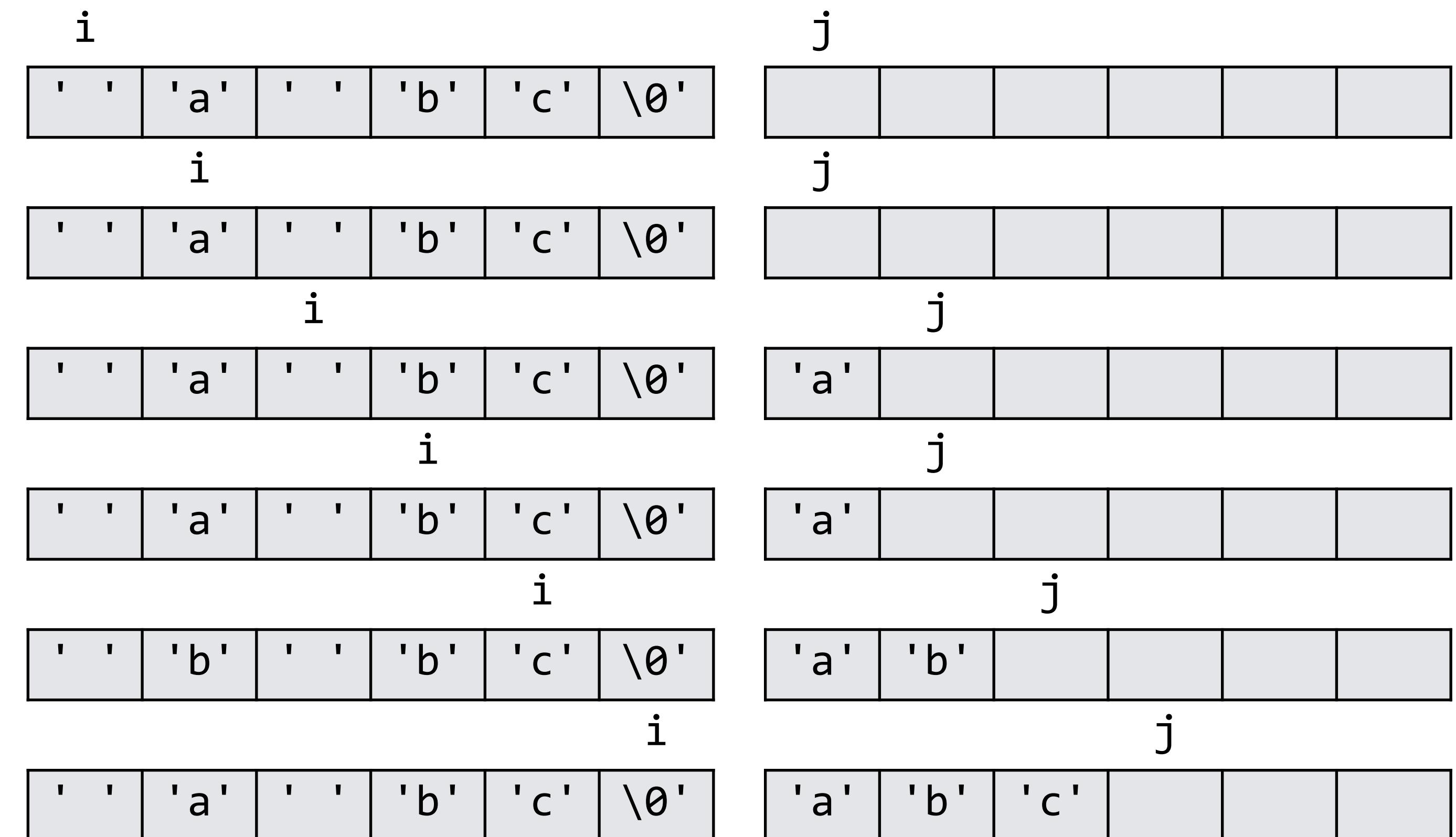
char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

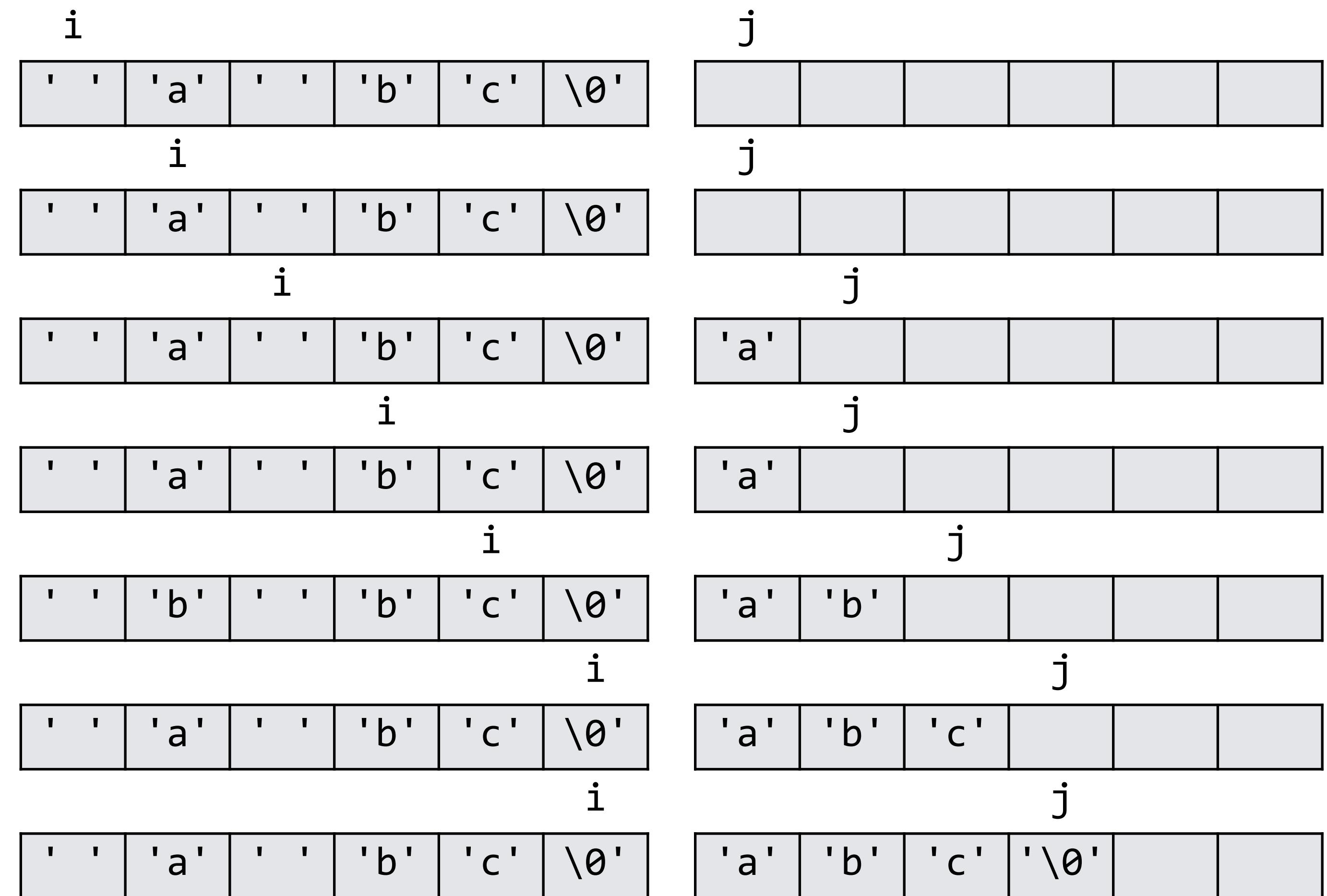
char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```



Remover espaços

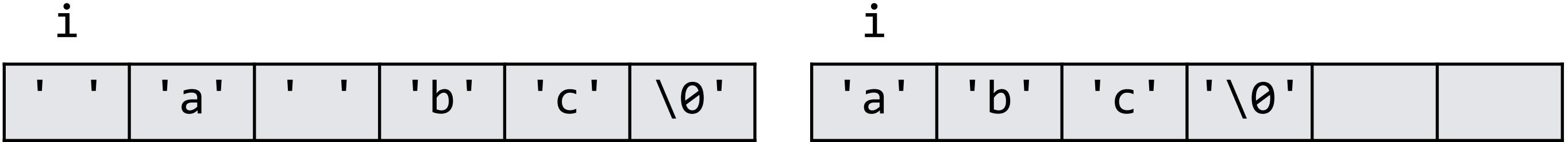
```
#include <ctype.h>

char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```

Remover espaços

```
#include <ctype.h>

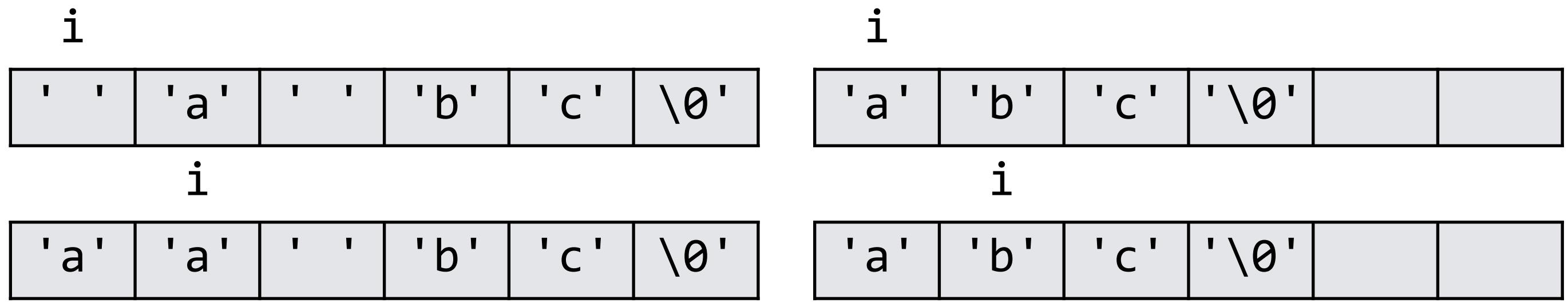
char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

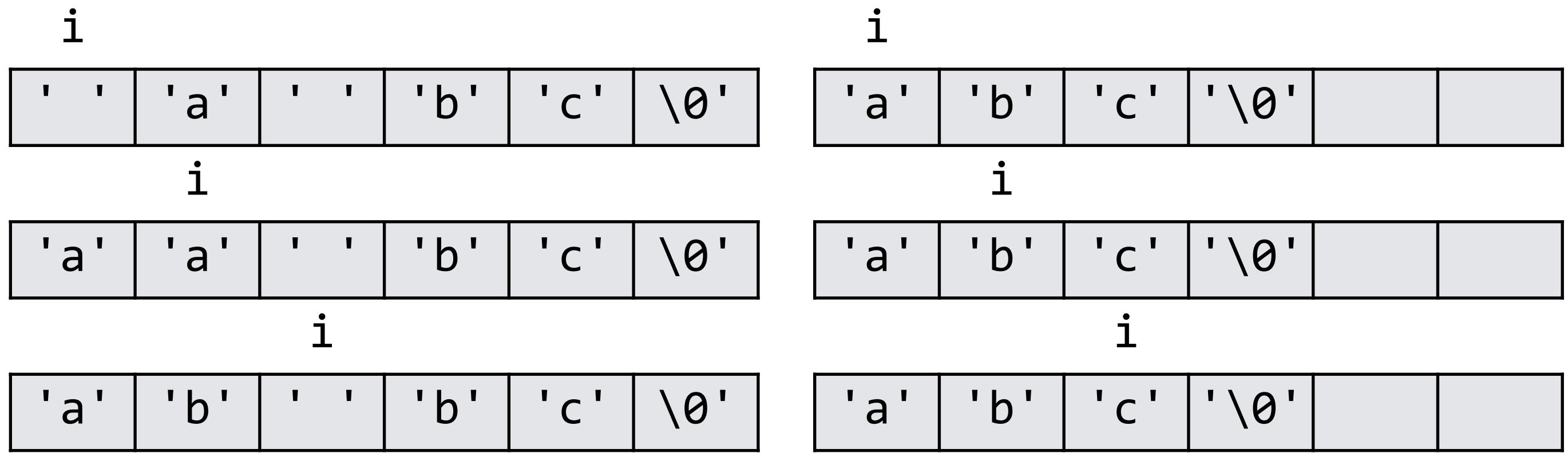
char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

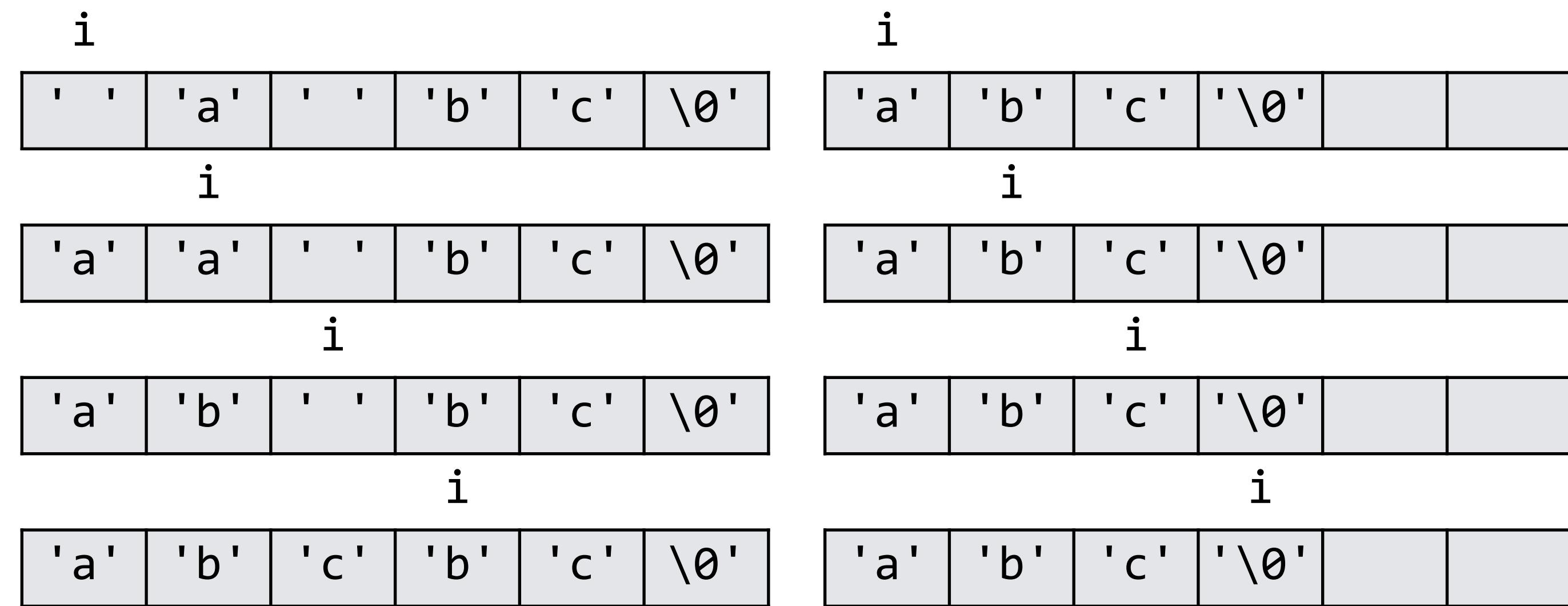
char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

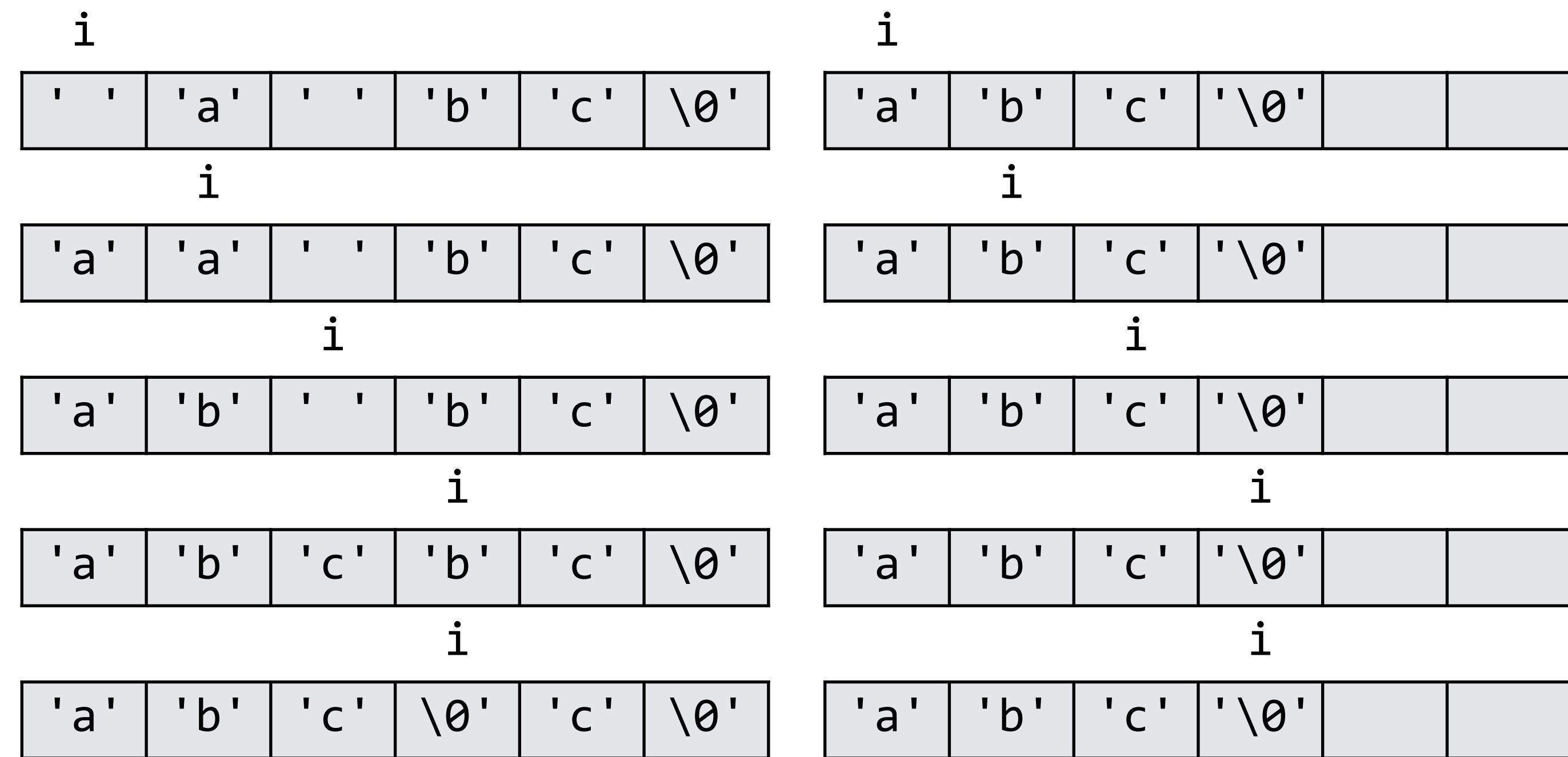
char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

char *trim(char s[]) {
    int i, j = 0;
    char t[strlen(s)+1];
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            t[j] = s[i]; j++;
        }
    t[j] = '\0';
    for (i = 0; t[i] != '\0'; i++)
        s[i] = t[i];
    s[i] = '\0';
    return s;
}
```



Remover espaços

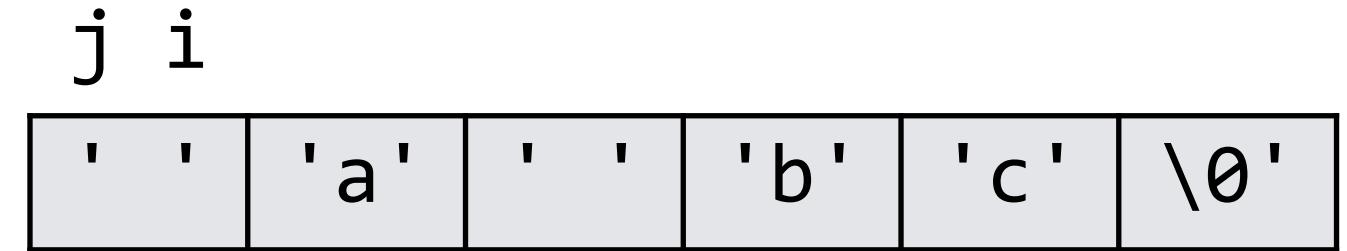
```
#include <ctype.h>

char *trim(char s[]) {
    int i, j = 0;
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            s[j] = s[i]; j++;
        }
    s[j] = '\0';
    return s;
}
```

Remover espaços

```
#include <ctype.h>

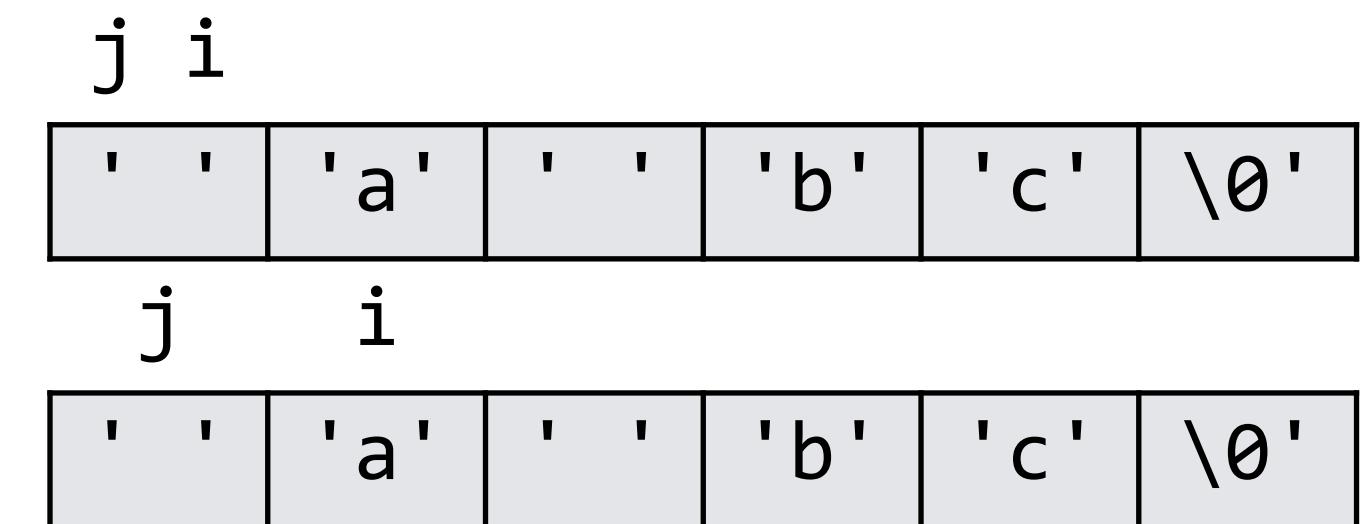
char *trim(char s[]) {
    int i, j = 0;
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            s[j] = s[i]; j++;
        }
    s[j] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

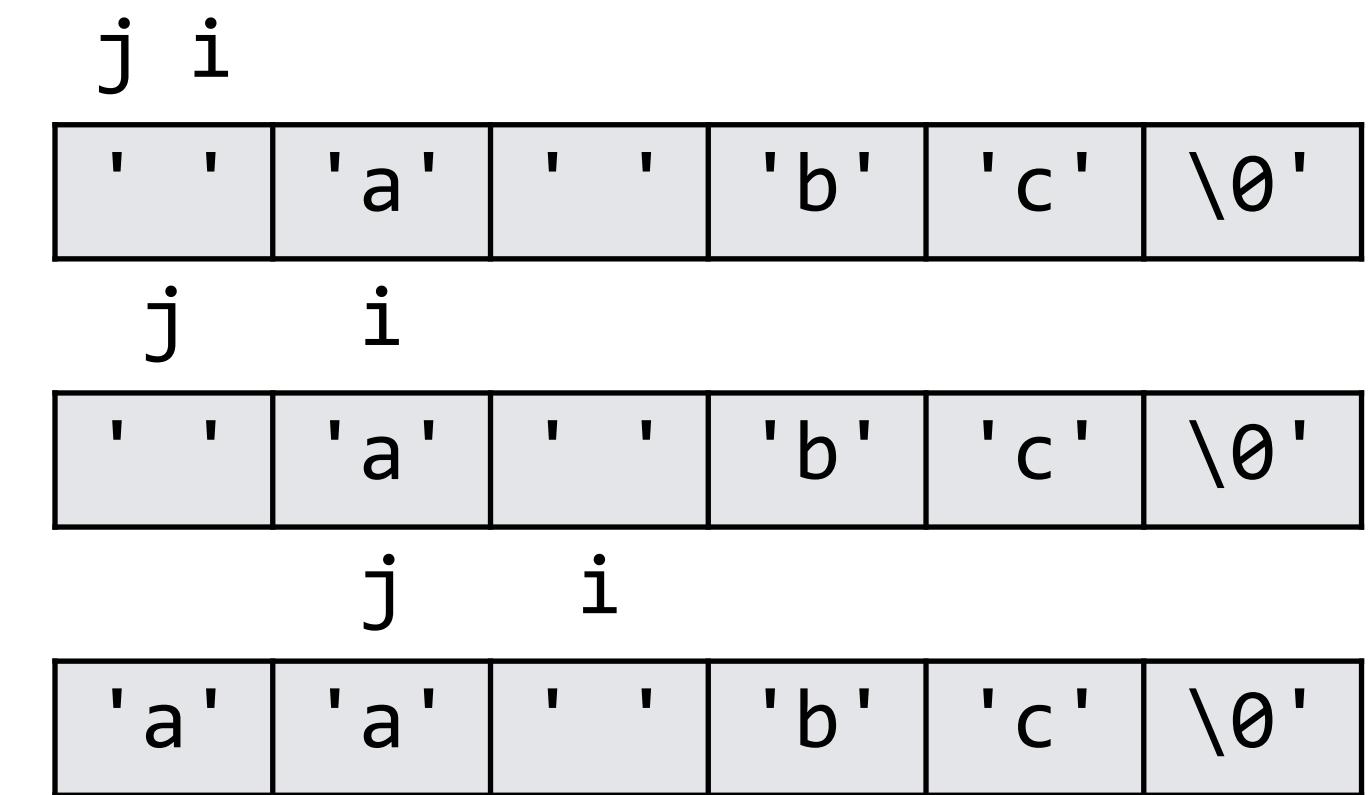
char *trim(char s[]) {
    int i, j = 0;
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            s[j] = s[i]; j++;
        }
    s[j] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

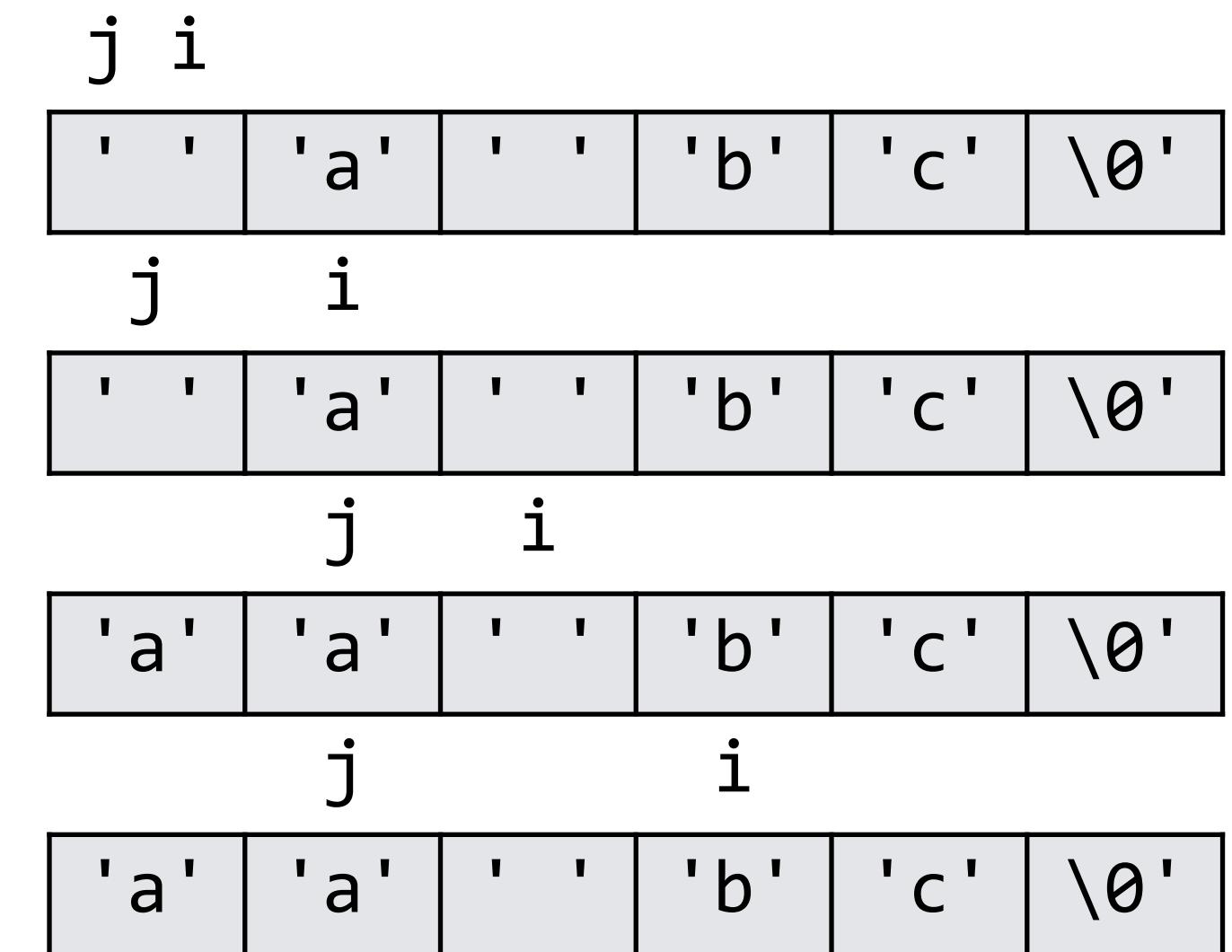
char *trim(char s[]) {
    int i, j = 0;
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            s[j] = s[i]; j++;
        }
    s[j] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

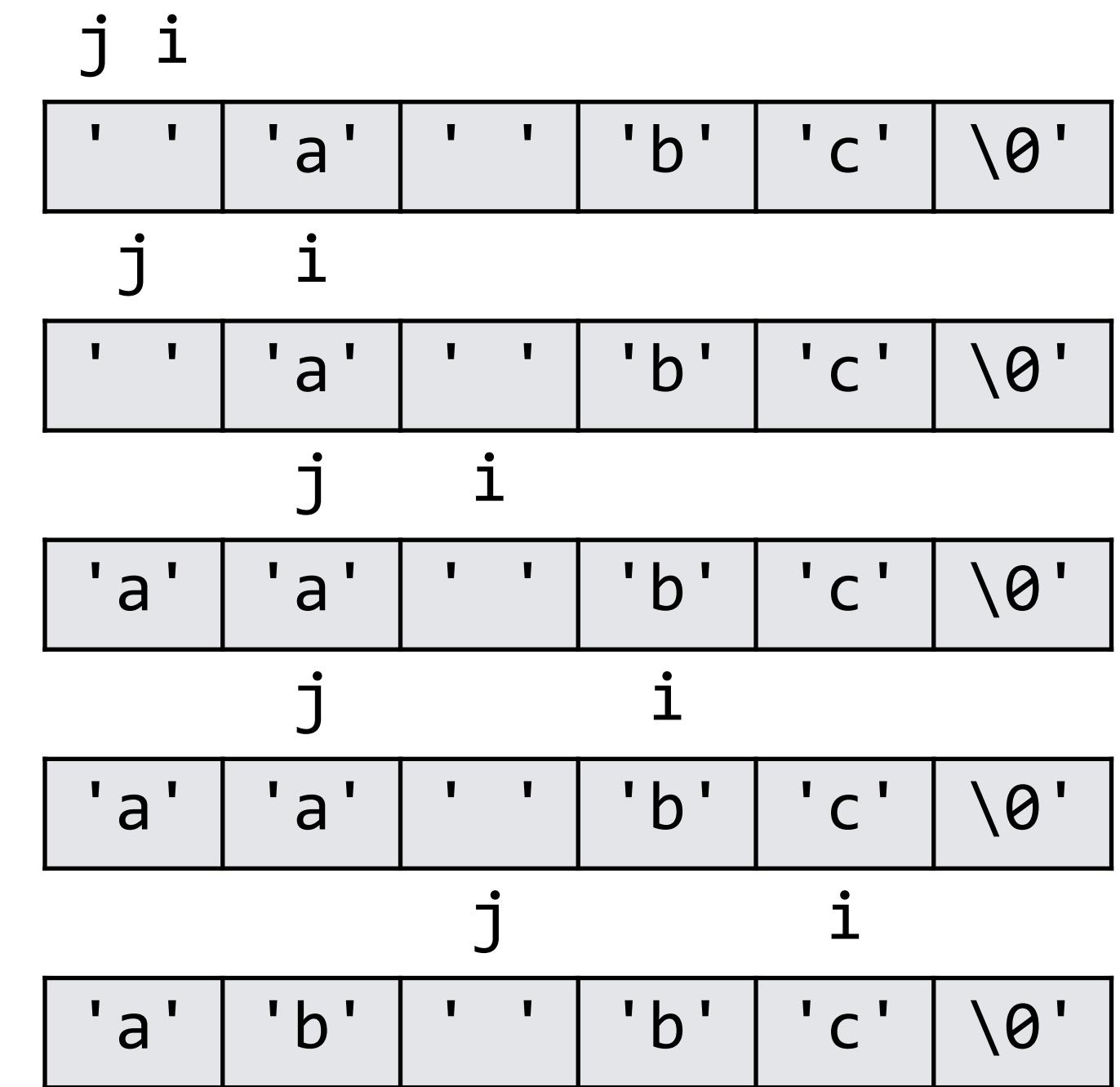
char *trim(char s[]) {
    int i, j = 0;
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            s[j] = s[i]; j++;
        }
    s[j] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

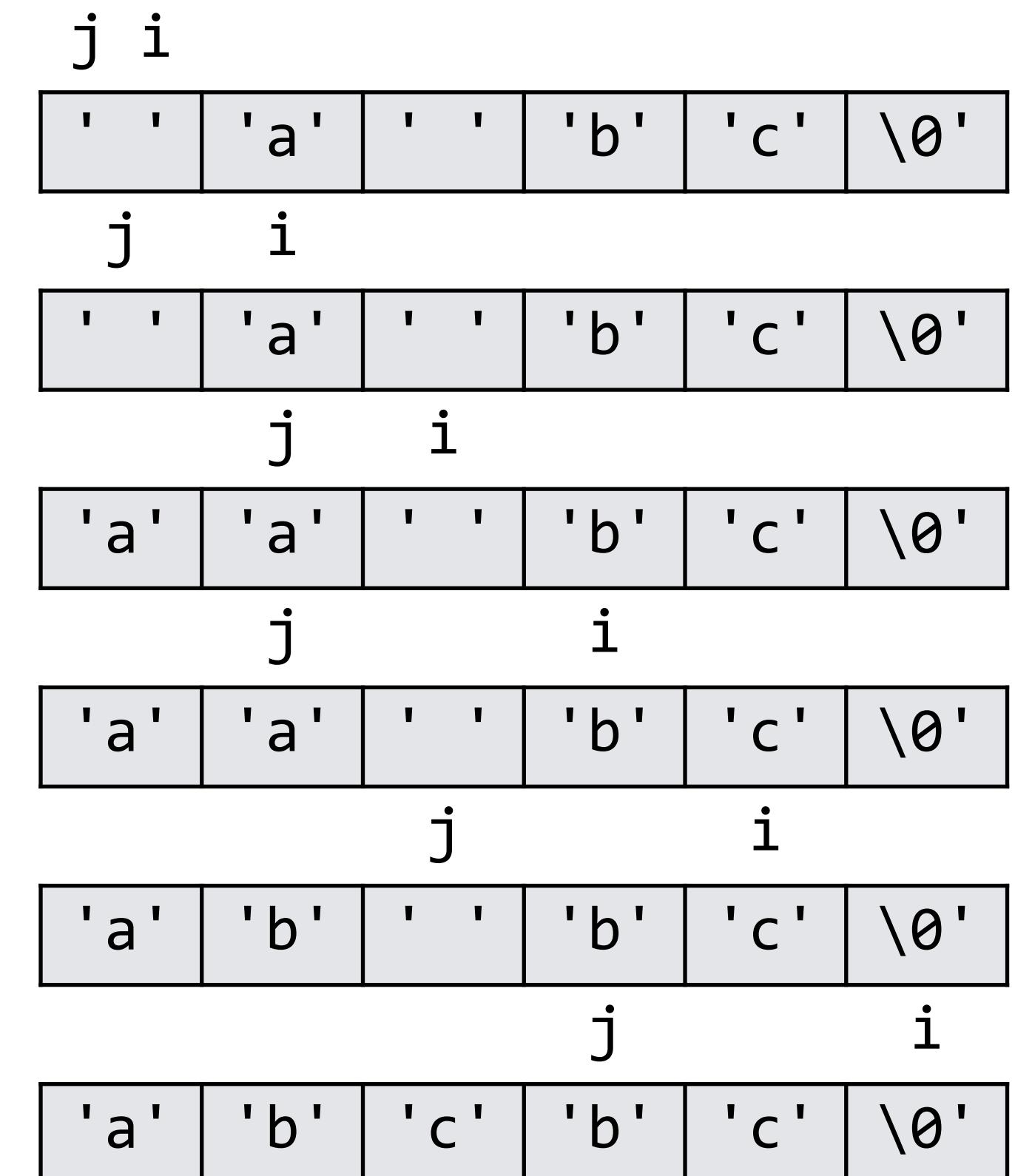
char *trim(char s[]) {
    int i, j = 0;
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            s[j] = s[i]; j++;
        }
    s[j] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

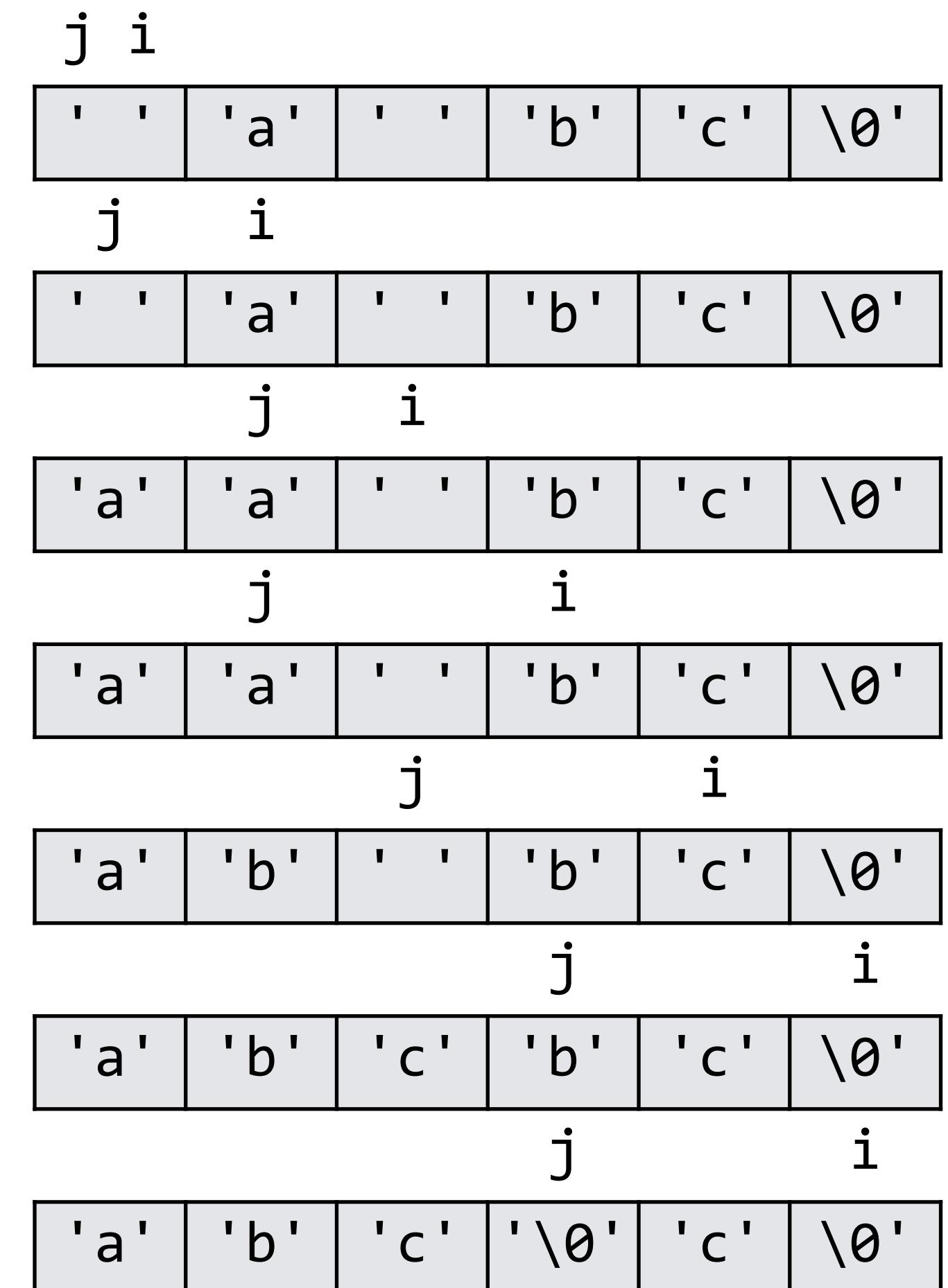
char *trim(char s[]) {
    int i, j = 0;
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            s[j] = s[i]; j++;
        }
    s[j] = '\0';
    return s;
}
```



Remover espaços

```
#include <ctype.h>

char *trim(char s[]) {
    int i, j = 0;
    for (i = 0; s[i] != '\0'; i++)
        if (!isspace(s[i])) {
            s[j] = s[i]; j++;
        }
    s[j] = '\0';
    return s;
}
```



gets

```
char *gets(char s[]) {  
    int i = 0;  
    while ((s[i] = getchar()) != '\n') i++;  
    s[i] = '\0';  
    return s;  
}
```

strcat

```
char *strcat(char s[], char t[]) {  
    int i = strlen(s), j = 0;  
    while (t[j] != '\0') {  
        s[i] = t[j];  
        i++; j++;  
    }  
    s[i] = '\0';  
    return s;  
}
```

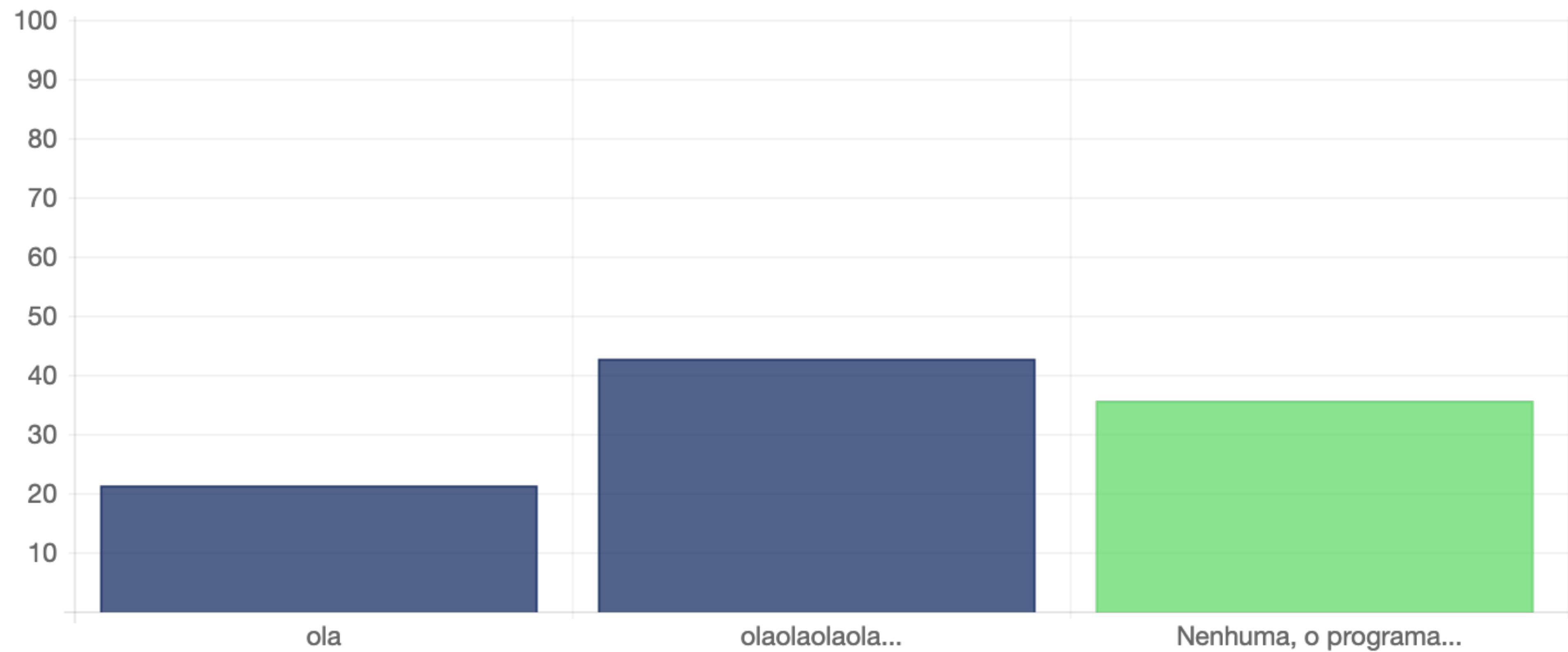
#13 Que imprime o seguinte programa?

```
char *strcat(char s[], char t[]) {  
    int i = strlen(s), j = 0;  
    while (t[j] != '\0') {  
        s[i] = t[j];  
        i++; j++;  
    }  
    s[i] = '\0';  
    return s;  
}
```

```
int main() {  
    char s[10] = "ola";  
    printf("%s\n", strcat(s,s));  
    return 0;  
}
```



#13 Que imprime o seguinte programa?



Aula 8

Pesquisa linear num array

```
int search(int x, int a[], int n) {  
    int i;  
    for (i = 0; i < n && a[i] != x; i++);  
    if (i < n) return 1;  
    else return 0;  
}
```

Sortear N inteiros diferentes

```
#define N 10

int main() {
    int i = 0, x, p[N];
    while (i < N) {
        x = rand();
        if (search(x,p,i)) continue;
        printf("%d\n", x);
        p[i] = x;
        i++;
    }
    return 0;
}
```



Pesquisa linear num array ordenado

```
int search(int x, int a[], int n) {  
    int i;  
    for (i = 0; i < n && a[i] < x; i++);  
    if (i < n && a[i] == x) return 1;  
    else return 0;  
}
```

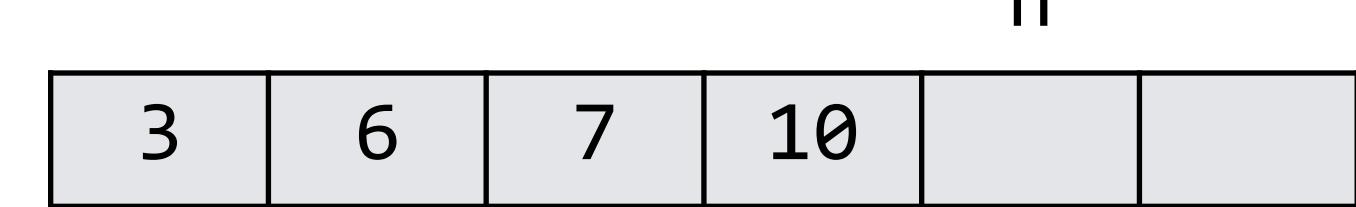
Inserção ordenada

```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}
```

```
int main() {  
    int a[6] = {3,6,7,10};  
    insert(5,a,4);  
    return 0;  
}
```

Inserção ordenada

```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}
```

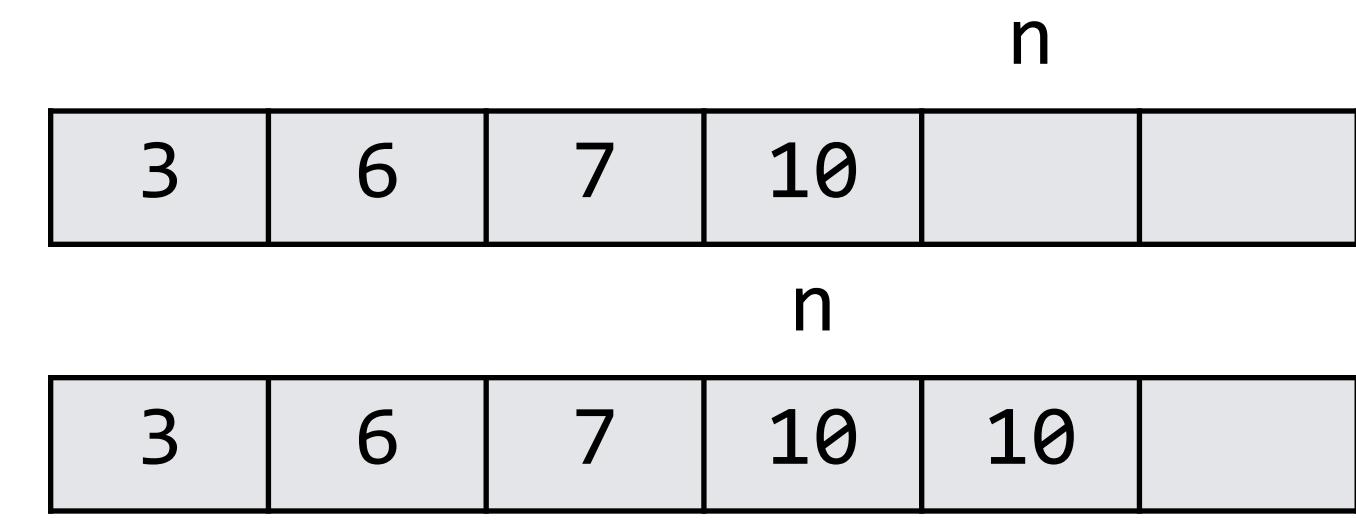


```
int main() {  
    int a[6] = {3,6,7,10};  
    insert(5,a,4);  
    return 0;  
}
```

Inserção ordenada

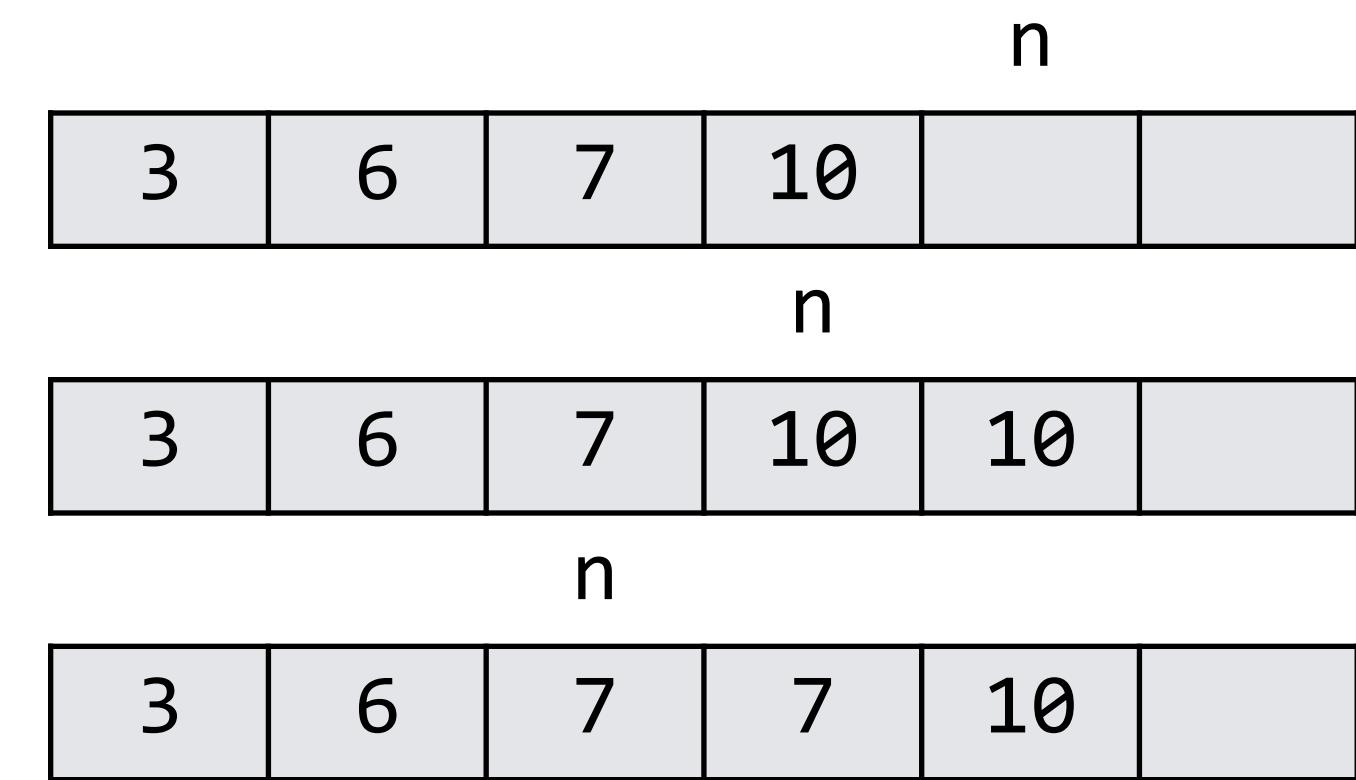
```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}
```

```
int main() {  
    int a[6] = {3,6,7,10};  
    insert(5,a,4);  
    return 0;  
}
```



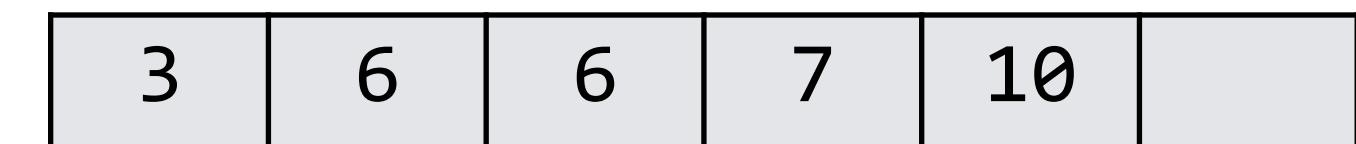
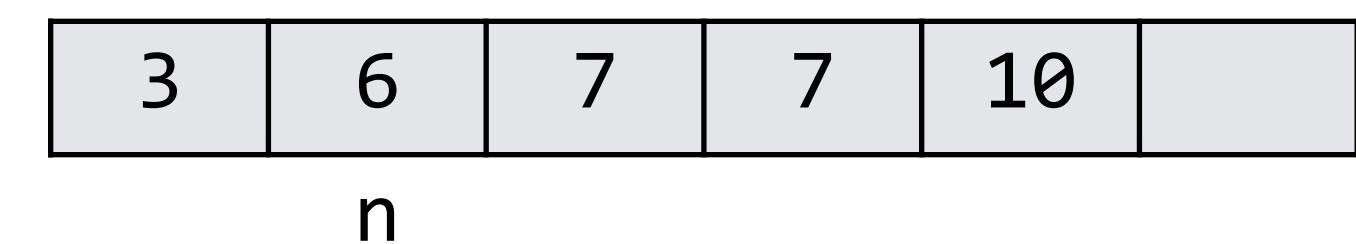
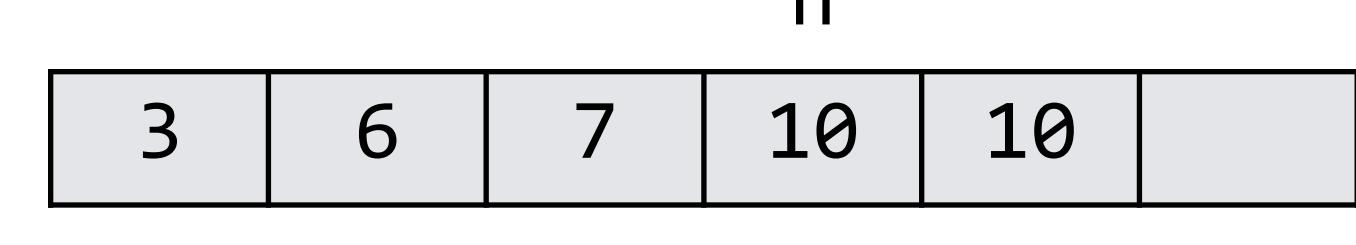
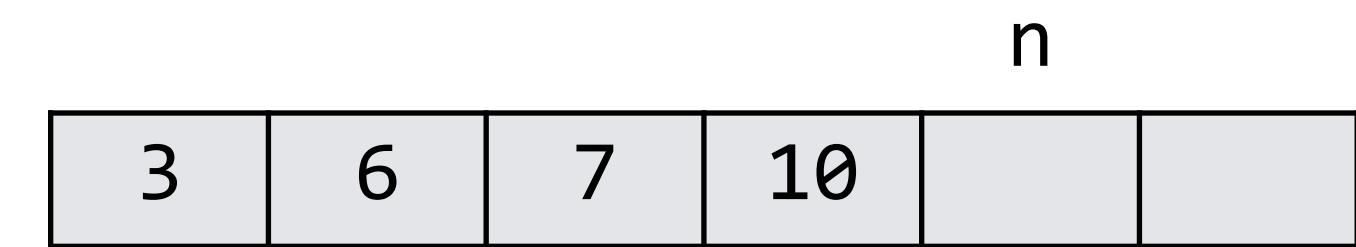
Inserção ordenada

```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}  
  
int main() {  
    int a[6] = {3,6,7,10};  
    insert(5,a,4);  
    return 0;  
}
```



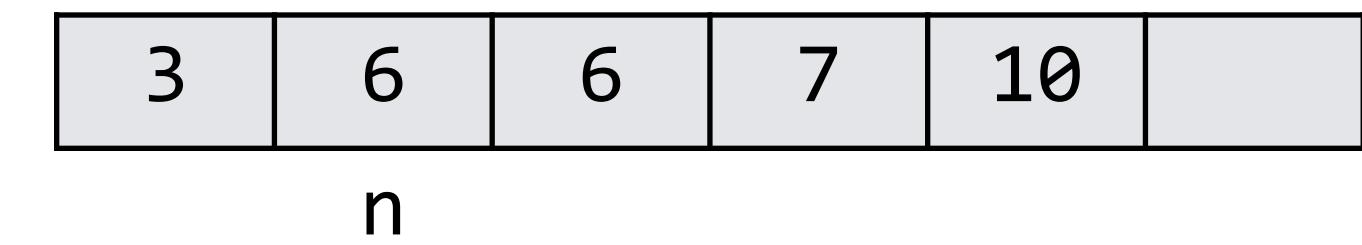
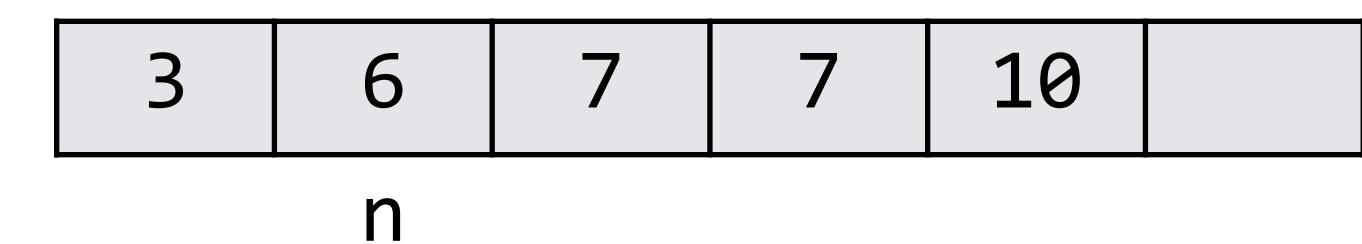
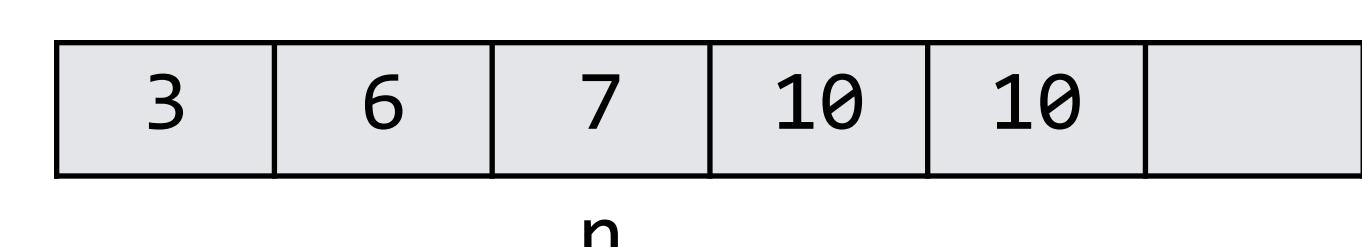
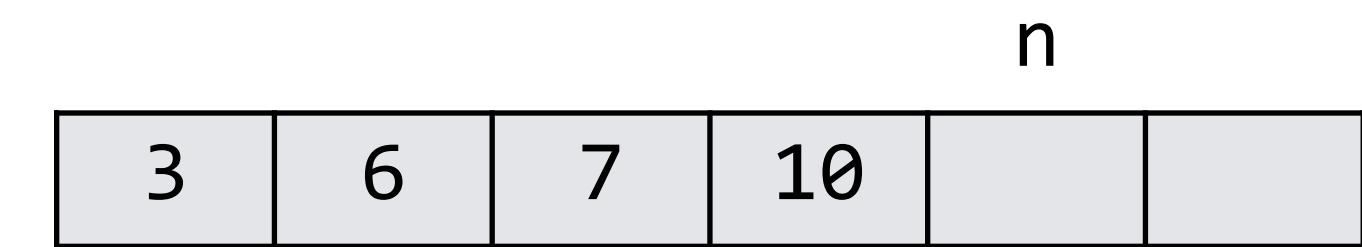
Inserção ordenada

```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}  
  
int main() {  
    int a[6] = {3,6,7,10};  
    insert(5,a,4);  
    return 0;  
}
```



Inserção ordenada

```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}  
  
int main() {  
    int a[6] = {3,6,7,10};  
    insert(5,a,4);  
    return 0;  
}
```



Sortear N inteiros diferentes

```
#define N 10

int main() {
    int i = 0, x, p[N];
    while (i < N) {
        x = rand();
        if (search(x,p,i)) continue;
        printf("%d\n", x);
        insert(x,p,i);
        i++;
    }
    return 0;
}
```



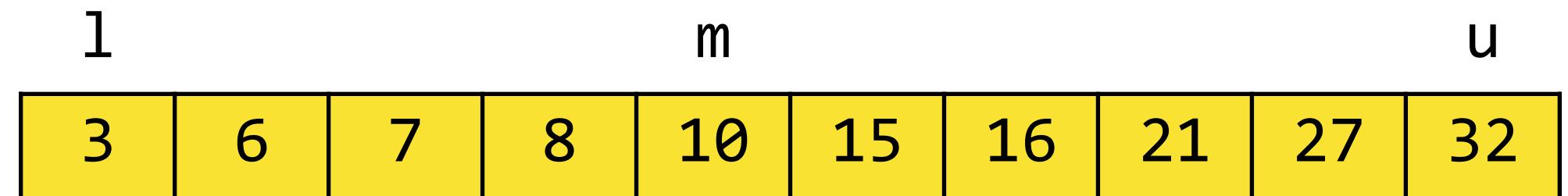
Pesquisa binária

```
int search(int x, int a[], int n) {
    int l = 0, u = n-1, m;
    while (l <= u) {
        m = (l+u) / 2;
        if (a[m] == x) return 1;
        if (a[m] < x) l = m+1;
        else u = m-1;
    }
    return 0;
}

int main() {
    int a[10] = {3,6,7,8,10,15,16,21,27,32};
    search(8,a,10);
    return 0;
}
```

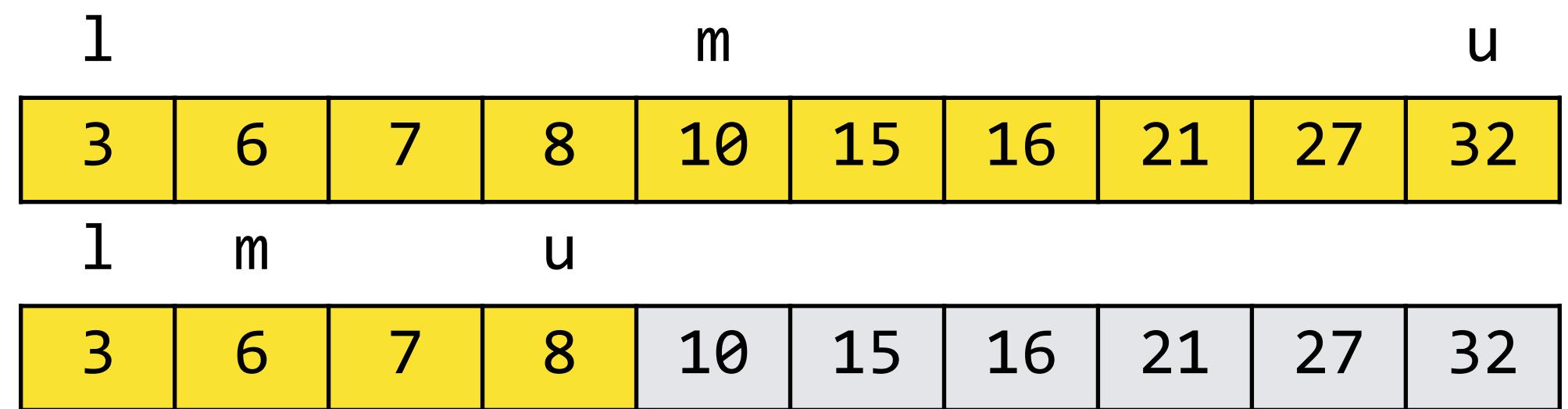
Pesquisa binária

```
int search(int x, int a[], int n) {  
    int l = 0, u = n-1, m;  
    while (l <= u) {  
        m = (l+u) / 2;  
        if (a[m] == x) return 1;  
        if (a[m] < x) l = m+1;  
        else u = m-1;  
    }  
    return 0;  
}  
  
int main() {  
    int a[10] = {3,6,7,8,10,15,16,21,27,32};  
    search(8,a,10);  
    return 0;  
}
```



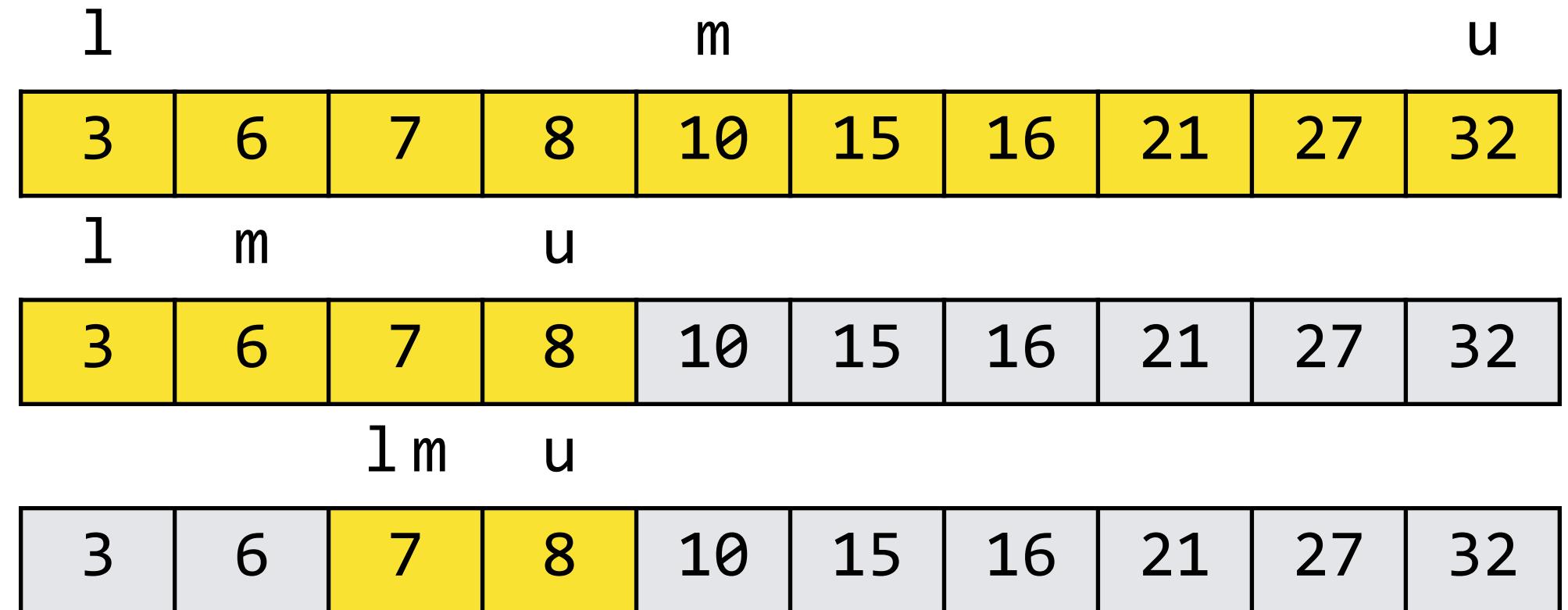
Pesquisa binária

```
int search(int x, int a[], int n) {  
    int l = 0, u = n-1, m;  
    while (l <= u) {  
        m = (l+u) / 2;  
        if (a[m] == x) return 1;  
        if (a[m] < x) l = m+1;  
        else u = m-1;  
    }  
    return 0;  
}  
  
int main() {  
    int a[10] = {3,6,7,8,10,15,16,21,27,32};  
    search(8,a,10);  
    return 0;  
}
```



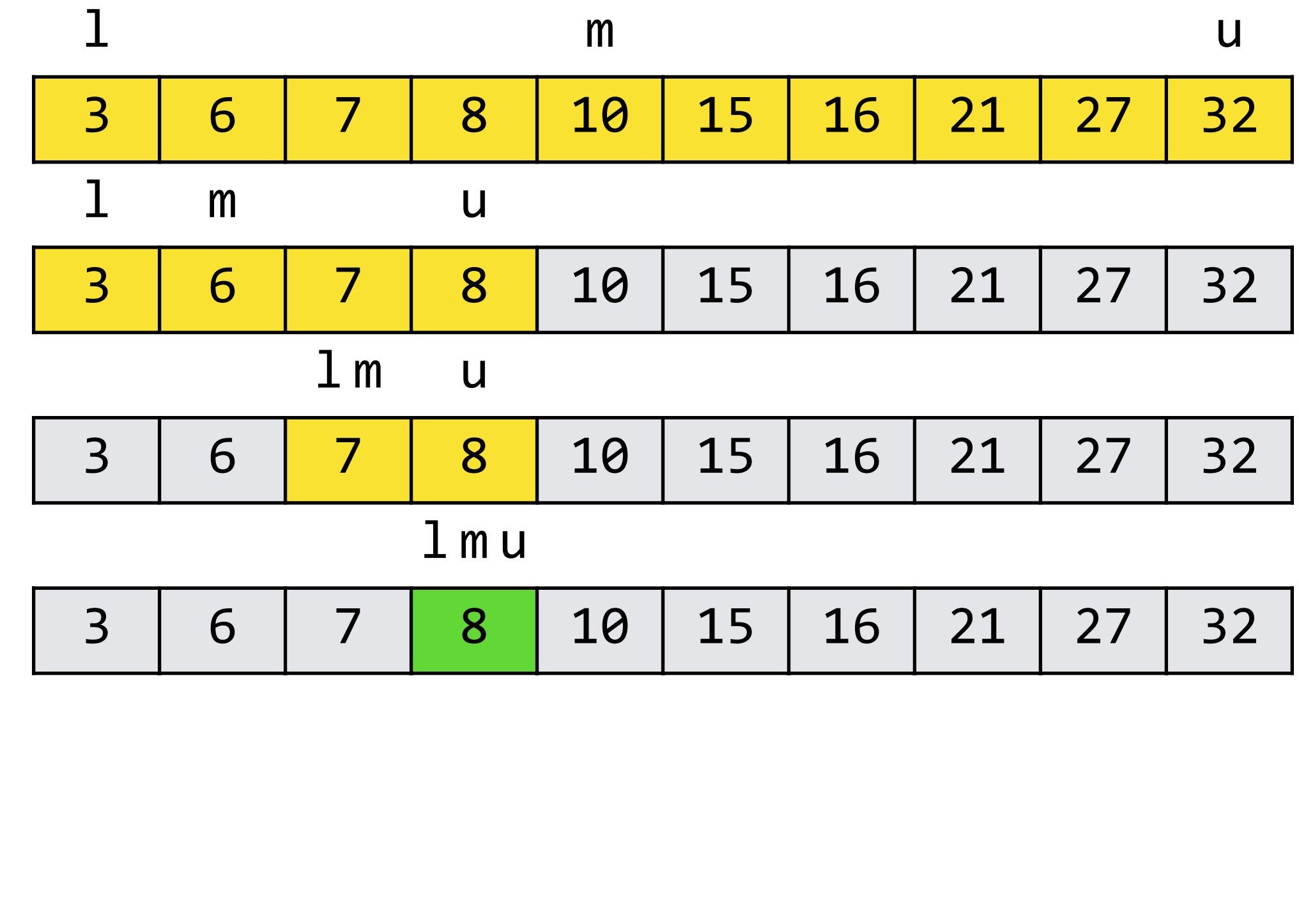
Pesquisa binária

```
int search(int x, int a[], int n) {  
    int l = 0, u = n-1, m;  
    while (l <= u) {  
        m = (l+u) / 2;  
        if (a[m] == x) return 1;  
        if (a[m] < x) l = m+1;  
        else u = m-1;  
    }  
    return 0;  
}  
  
int main() {  
    int a[10] = {3,6,7,8,10,15,16,21,27,32};  
    search(8,a,10);  
    return 0;  
}
```



Pesquisa binária

```
int search(int x, int a[], int n) {  
    int l = 0, u = n-1, m;  
    while (l <= u) {  
        m = (l+u) / 2;  
        if (a[m] == x) return 1;  
        if (a[m] < x) l = m+1;  
        else u = m-1;  
    }  
    return 0;  
}  
  
int main() {  
    int a[10] = {3,6,7,8,10,15,16,21,27,32};  
    search(8,a,10);  
    return 0;  
}
```



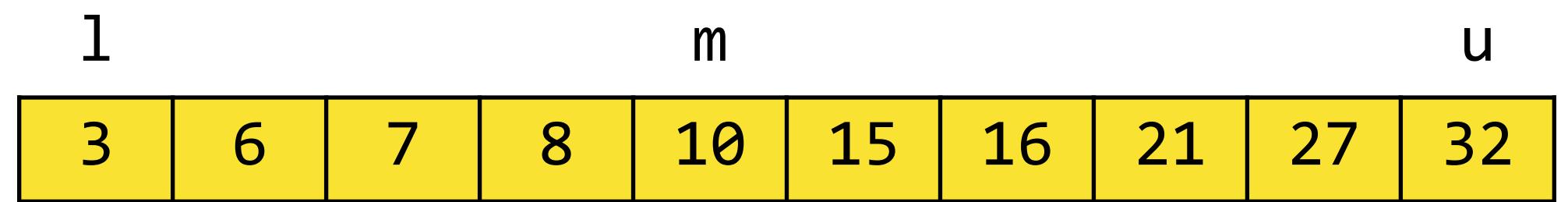
Pesquisa binária

```
int search(int x, int a[], int n) {
    int l = 0, u = n-1, m;
    while (l <= u) {
        m = (l+u) / 2;
        if (a[m] == x) return 1;
        if (a[m] < x) l = m+1;
        else u = m-1;
    }
    return 0;
}

int main() {
    int a[10] = {3,6,7,8,10,15,16,21,27,32};
    search(20,a,10);
    return 0;
}
```

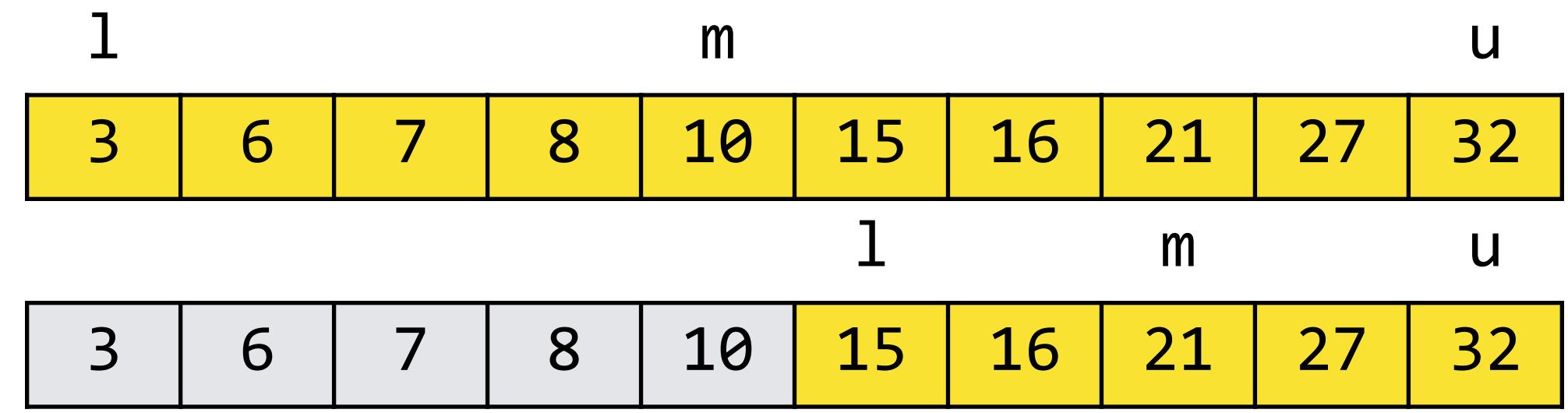
Pesquisa binária

```
int search(int x, int a[], int n) {  
    int l = 0, u = n-1, m;  
    while (l <= u) {  
        m = (l+u) / 2;  
        if (a[m] == x) return 1;  
        if (a[m] < x) l = m+1;  
        else u = m-1;  
    }  
    return 0;  
}  
  
int main() {  
    int a[10] = {3,6,7,8,10,15,16,21,27,32};  
    search(20,a,10);  
    return 0;  
}
```



Pesquisa binária

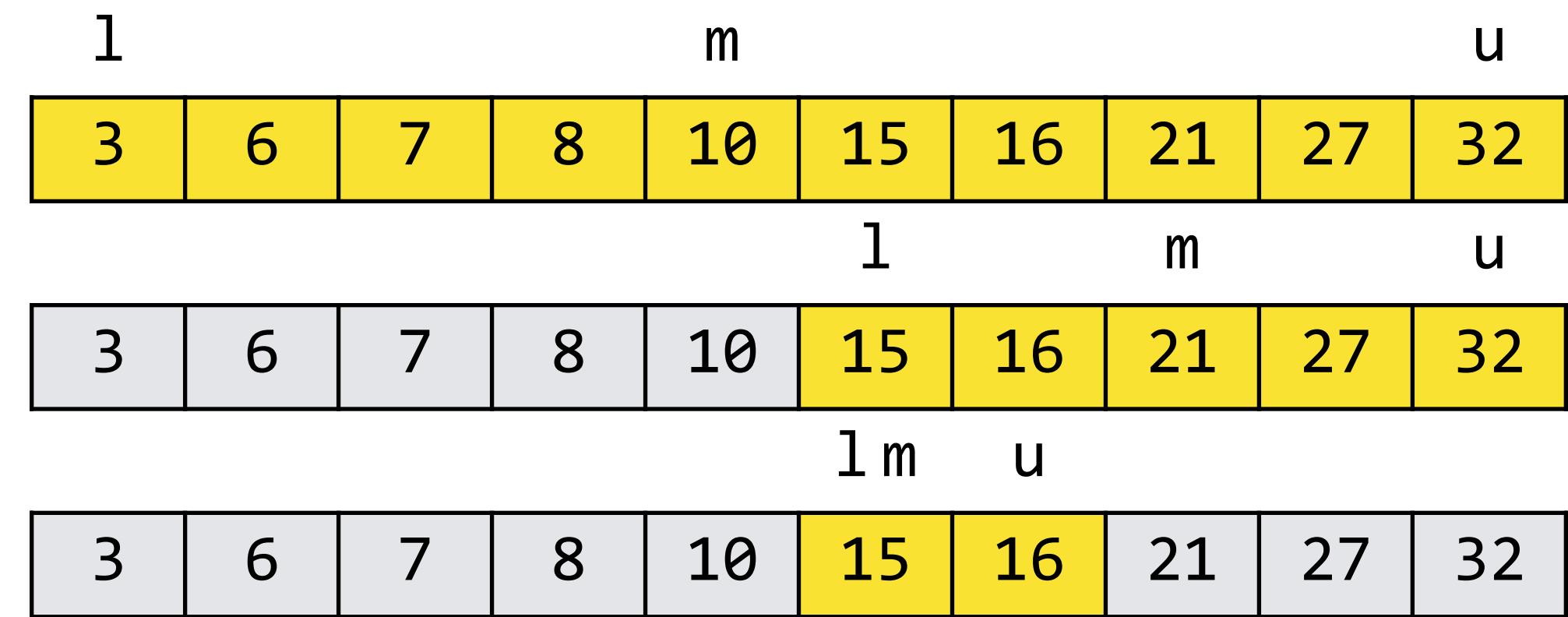
```
int search(int x, int a[], int n) {  
    int l = 0, u = n-1, m;  
    while (l <= u) {  
        m = (l+u) / 2;  
        if (a[m] == x) return 1;  
        if (a[m] < x) l = m+1;  
        else u = m-1;  
    }  
    return 0;  
}  
  
int main() {  
    int a[10] = {3,6,7,8,10,15,16,21,27,32};  
    search(20,a,10);  
    return 0;  
}
```



Pesquisa binária

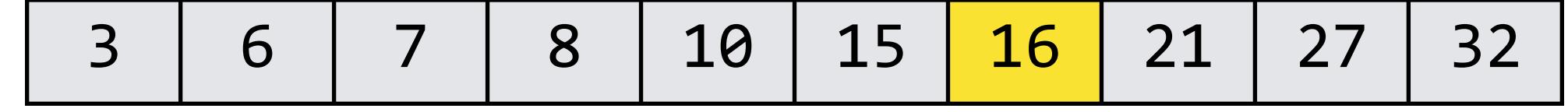
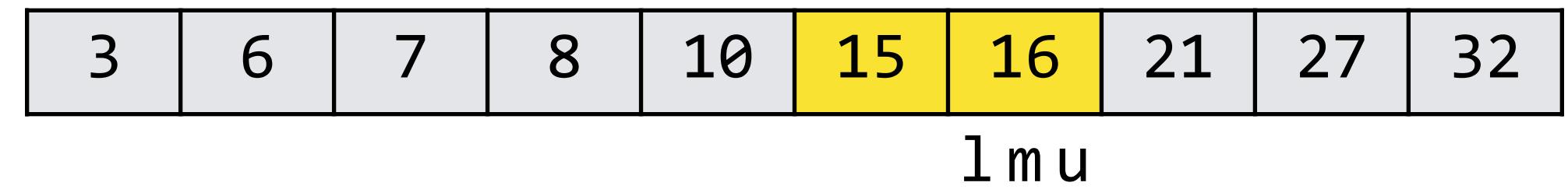
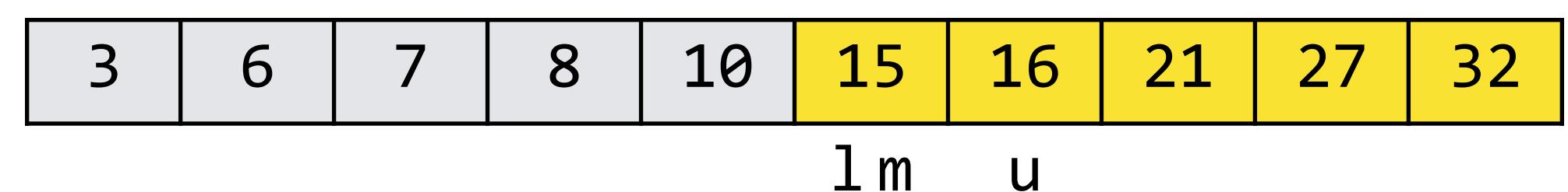
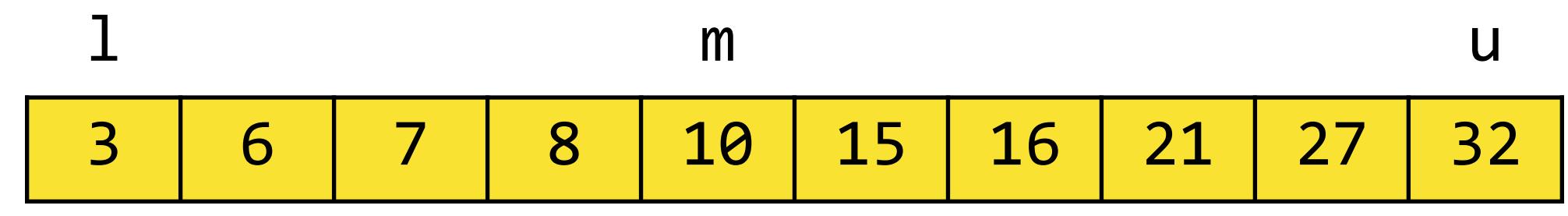
```
int search(int x, int a[], int n) {  
    int l = 0, u = n-1, m;  
    while (l <= u) {  
        m = (l+u) / 2;  
        if (a[m] == x) return 1;  
        if (a[m] < x) l = m+1;  
        else u = m-1;  
    }  
    return 0;  
}
```

```
int main() {  
    int a[10] = {3,6,7,8,10,15,16,21,27,32};  
    search(20,a,10);  
    return 0;  
}
```



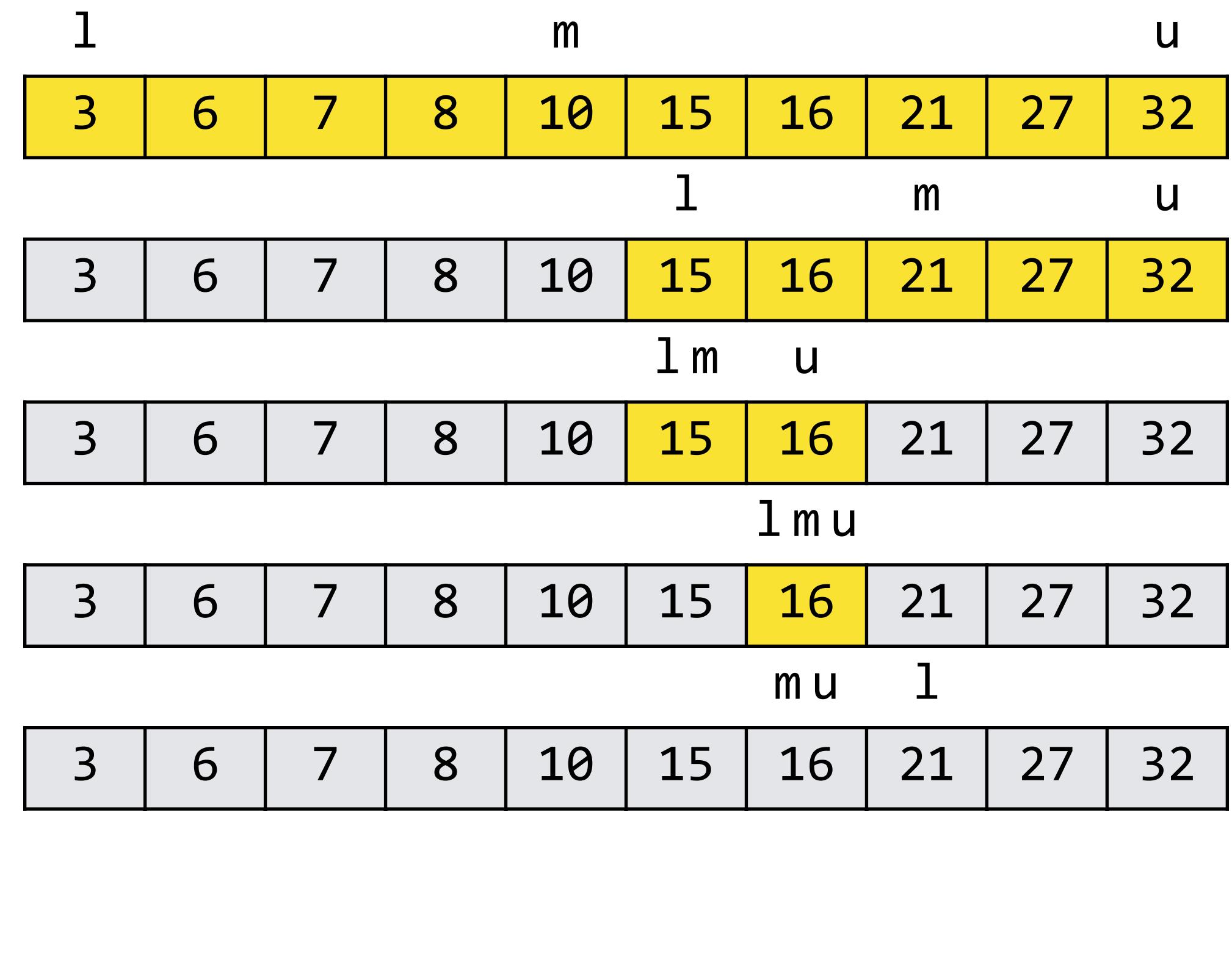
Pesquisa binária

```
int search(int x, int a[], int n) {  
    int l = 0, u = n-1, m;  
    while (l <= u) {  
        m = (l+u) / 2;  
        if (a[m] == x) return 1;  
        if (a[m] < x) l = m+1;  
        else u = m-1;  
    }  
    return 0;  
}  
  
int main() {  
    int a[10] = {3,6,7,8,10,15,16,21,27,32};  
    search(20,a,10);  
    return 0;  
}
```



Pesquisa binária

```
int search(int x, int a[], int n) {  
    int l = 0, u = n-1, m;  
    while (l <= u) {  
        m = (l+u) / 2;  
        if (a[m] == x) return 1;  
        if (a[m] < x) l = m+1;  
        else u = m-1;  
    }  
    return 0;  
}  
  
int main() {  
    int a[10] = {3,6,7,8,10,15,16,21,27,32};  
    search(20,a,10);  
    return 0;  
}
```



#14 Aproximadamente, quantas iterações fará o search?

```
#define N ...  
  
int main() {  
    int a[N];  
    for (int i = 0; i < N; i++) a[i] = i;  
    search(N,a,N);  
}
```



Pesquisa binária

N	\sqrt{N}	$\log_2(N)$
16	4	4
256	16	8
1024	32	10
1048576	1024	20



Insertion sort

```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}
```

```
void isort(int a[], int n) {  
    if (n == 0) return;  
    isort(a,n-1);  
    insert(a[n-1],a,n-1);  
}
```

Insertion sort

```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}
```

```
void isort(int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        insert(a[i],a,i);  
}
```

Insertion sort

```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}
```

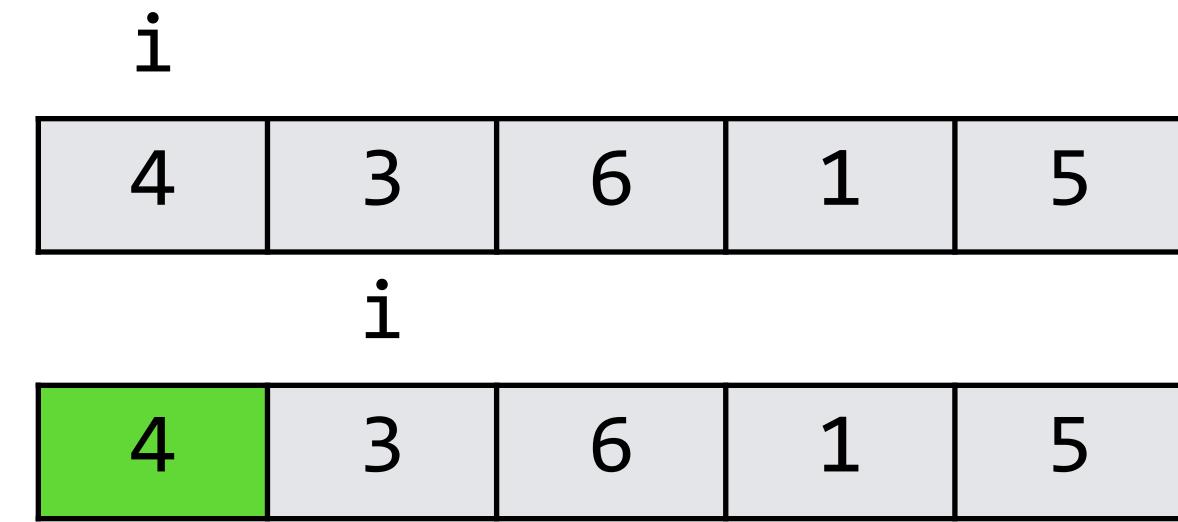
i	4	3	6	1	5
---	---	---	---	---	---

```
void isort(int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        insert(a[i], a, i);  
}
```

Insertion sort

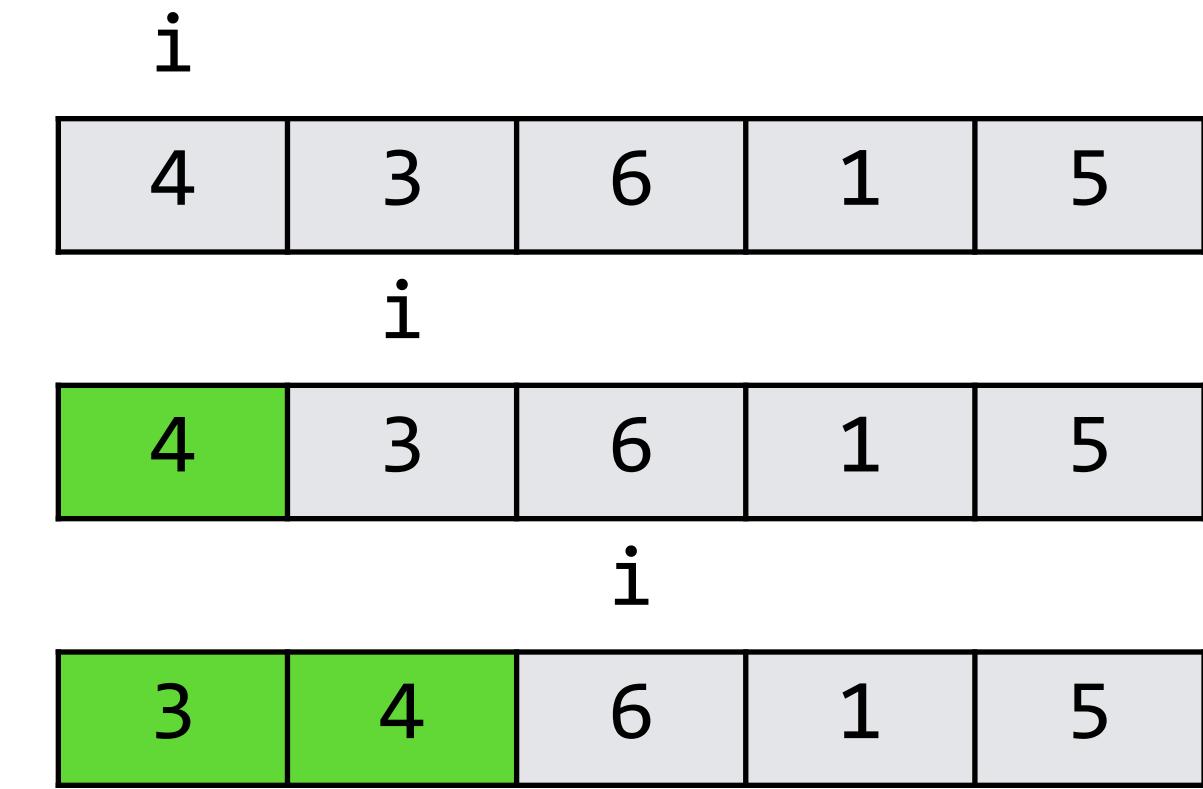
```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}
```

```
void isort(int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        insert(a[i], a, i);  
}
```



Insertion sort

```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}
```

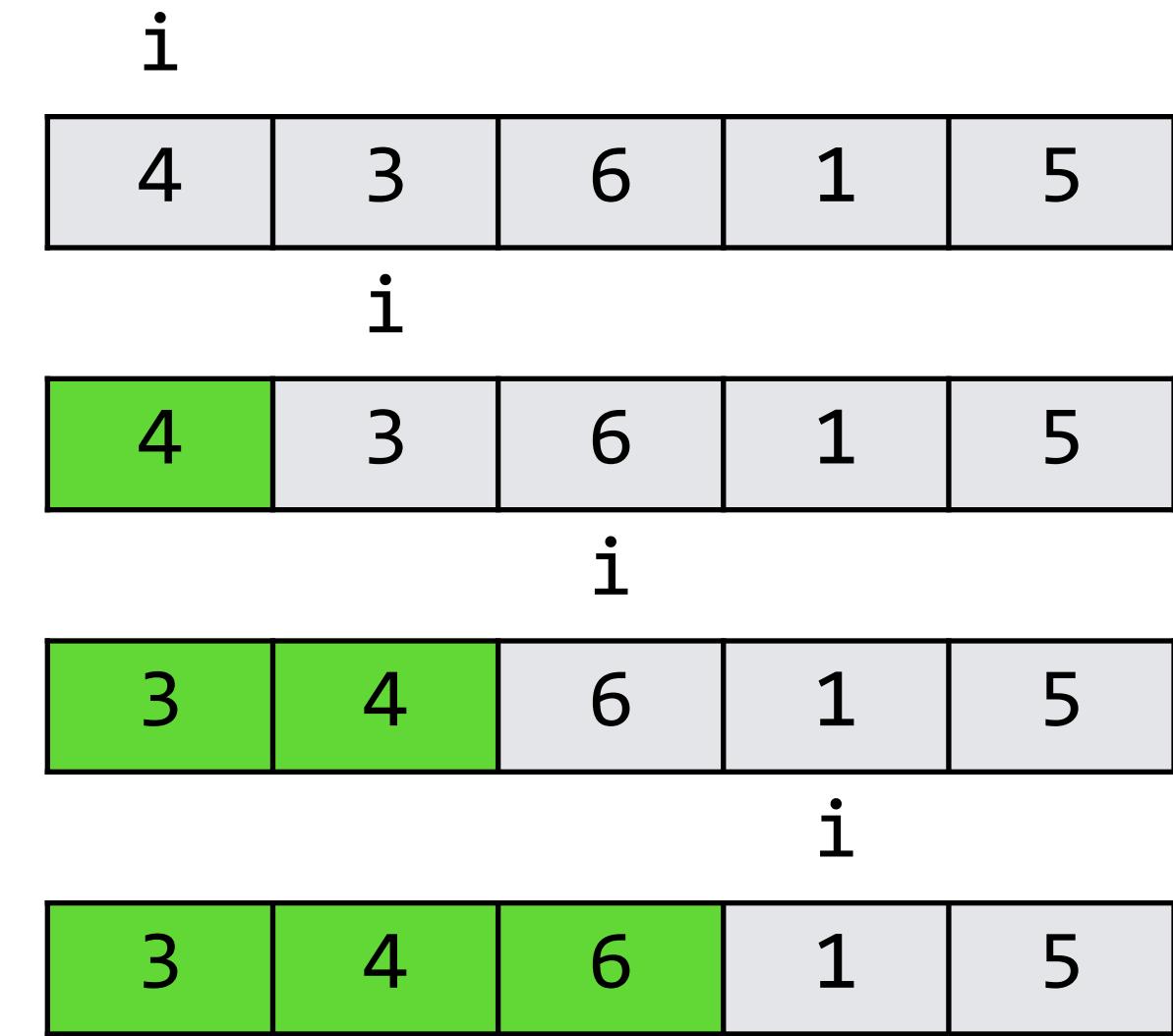


```
void isort(int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        insert(a[i], a, i);  
}
```

Insertion sort

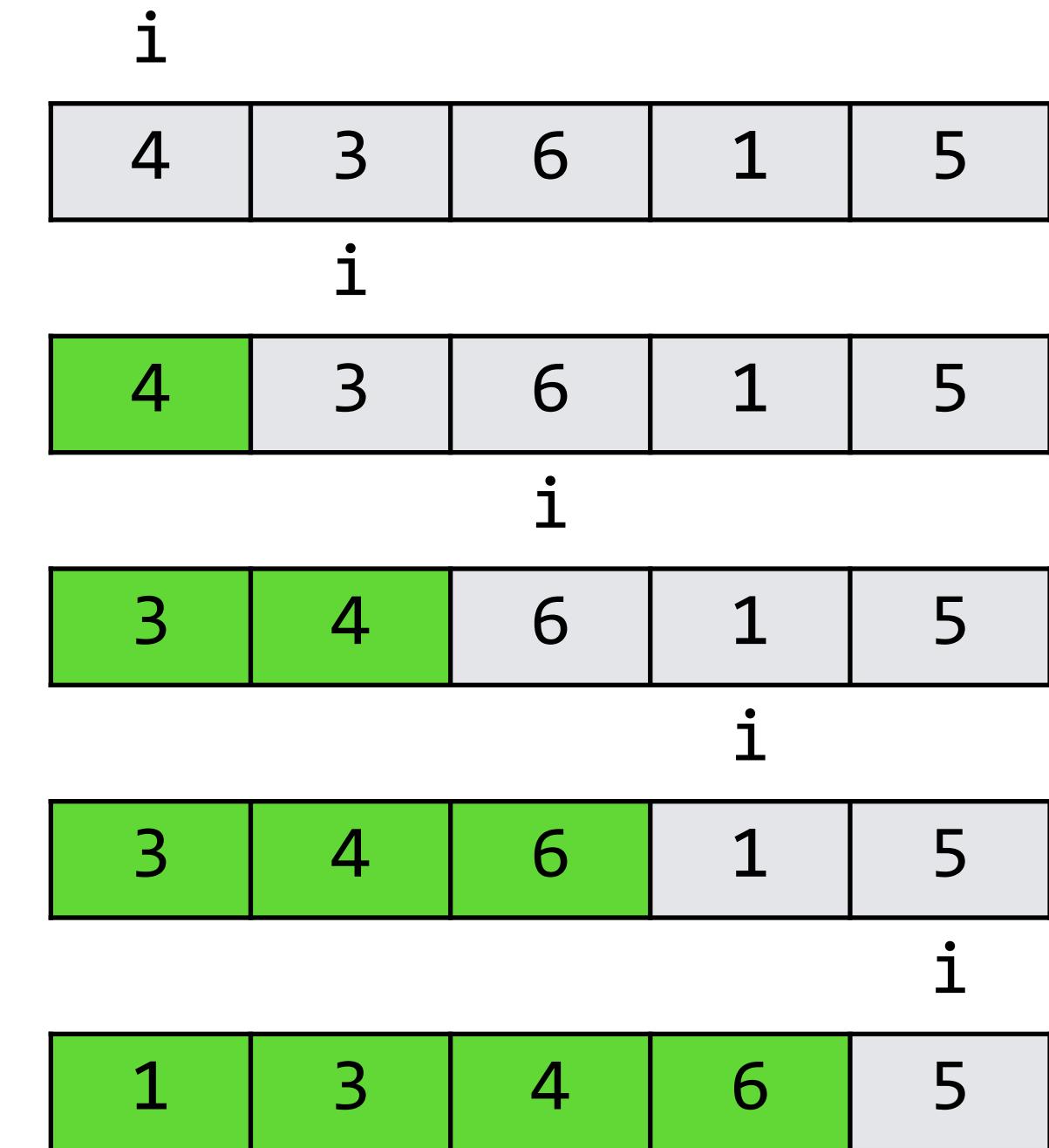
```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}
```

```
void isort(int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        insert(a[i], a, i);  
}
```



Insertion sort

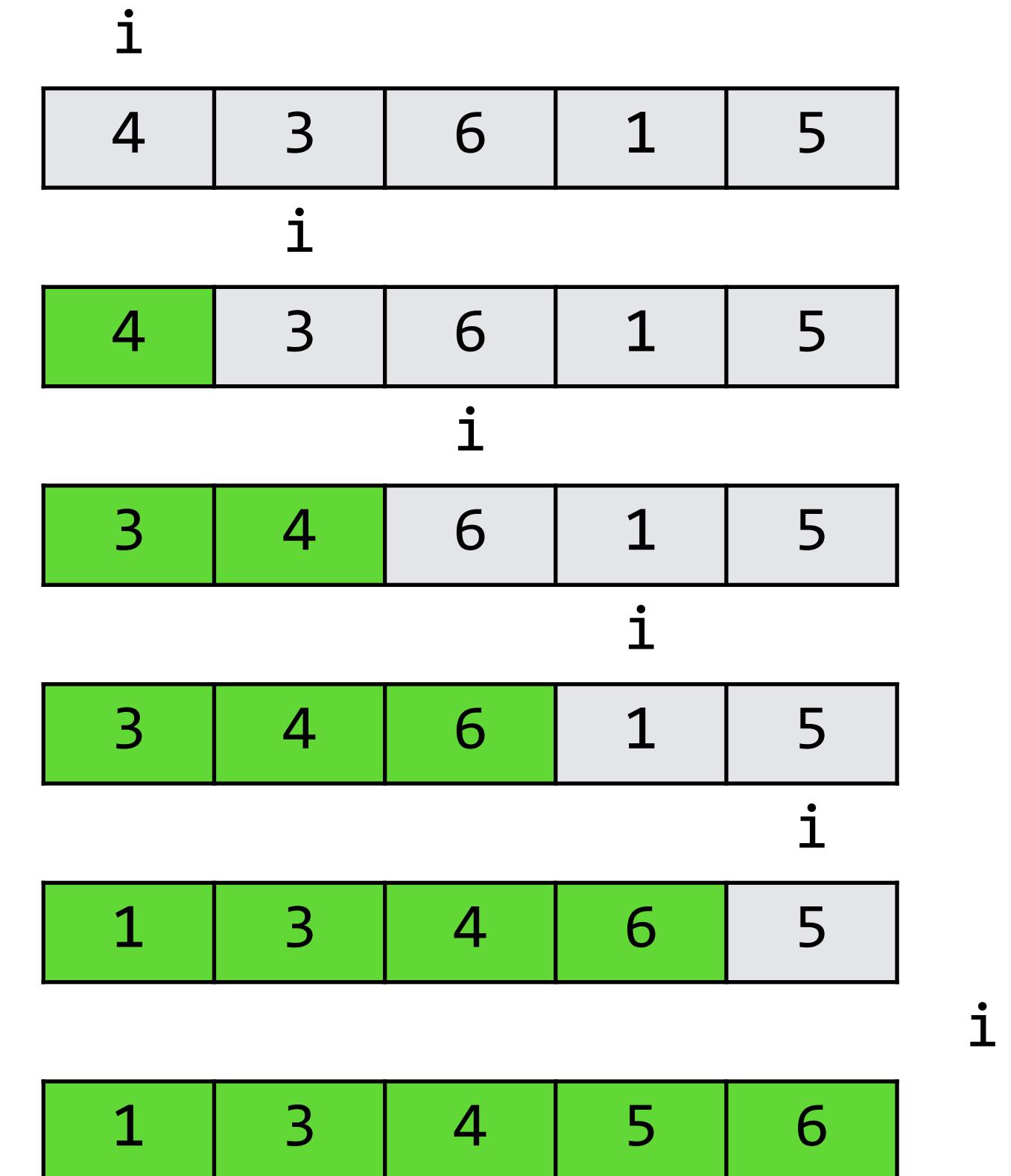
```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}  
  
void isort(int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        insert(a[i], a, i);  
}
```



Insertion sort

```
void insert(int x, int a[], int n) {  
    while (n > 0 && a[n-1] > x) {  
        a[n] = a[n-1];  
        n--;  
    }  
    a[n] = x;  
}
```

```
void isort(int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        insert(a[i], a, i);  
}
```



Insertion sort

```
void isort(int a[], int n) {  
    int i, j, aux;  
    for (i = 0; i < n; i++) {  
        aux = a[i];  
        for (j = i; j > 0 && a[j-1] > aux; j--) a[j] = a[j-1];  
        a[j] = aux;  
    }  
}
```

swap

```
void swap(int a[], int i, int j) {  
    int aux = a[i];  
    a[i] = a[j];  
    a[j] = aux;  
}
```

Selection sort

```
int max(int a[], int n) {  
    int m = 0, i;  
    for (i = 1; i < n; i++)  
        if (a[i] > a[m]) m = i;  
    return m;  
}
```

```
void ssort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        swap(a, i-1, max(a,i));  
}
```

Selection sort

```
int max(int a[], int n) {  
    int m = 0, i;  
    for (i = 1; i < n; i++)  
        if (a[i] > a[m]) m = i;  
    return m;  
}
```

4	3	6	1	5
---	---	---	---	---

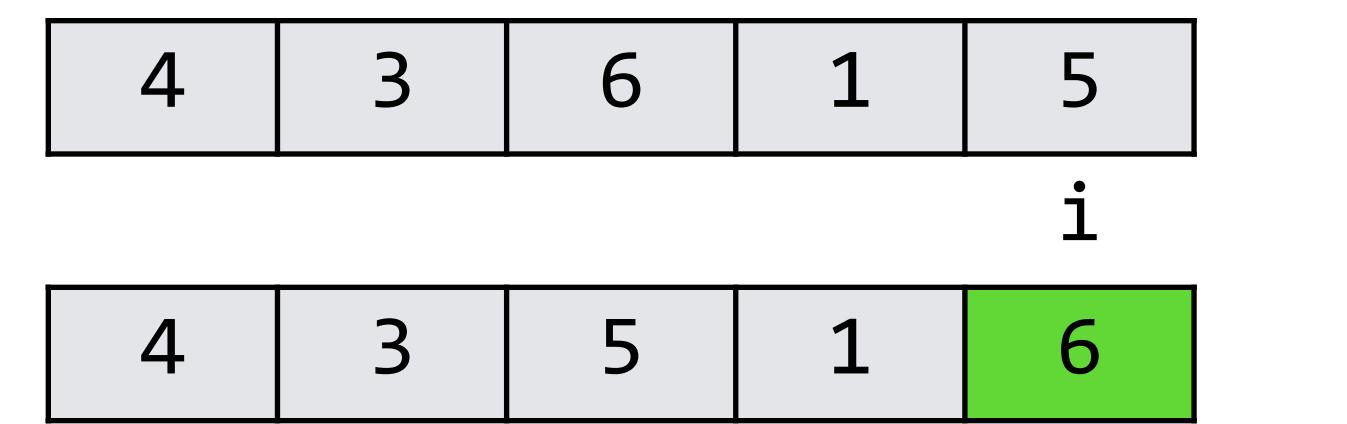
i

```
void ssort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        swap(a, i-1, max(a,i));  
}
```

Selection sort

```
int max(int a[], int n) {  
    int m = 0, i;  
    for (i = 1; i < n; i++)  
        if (a[i] > a[m]) m = i;  
    return m;  
}
```

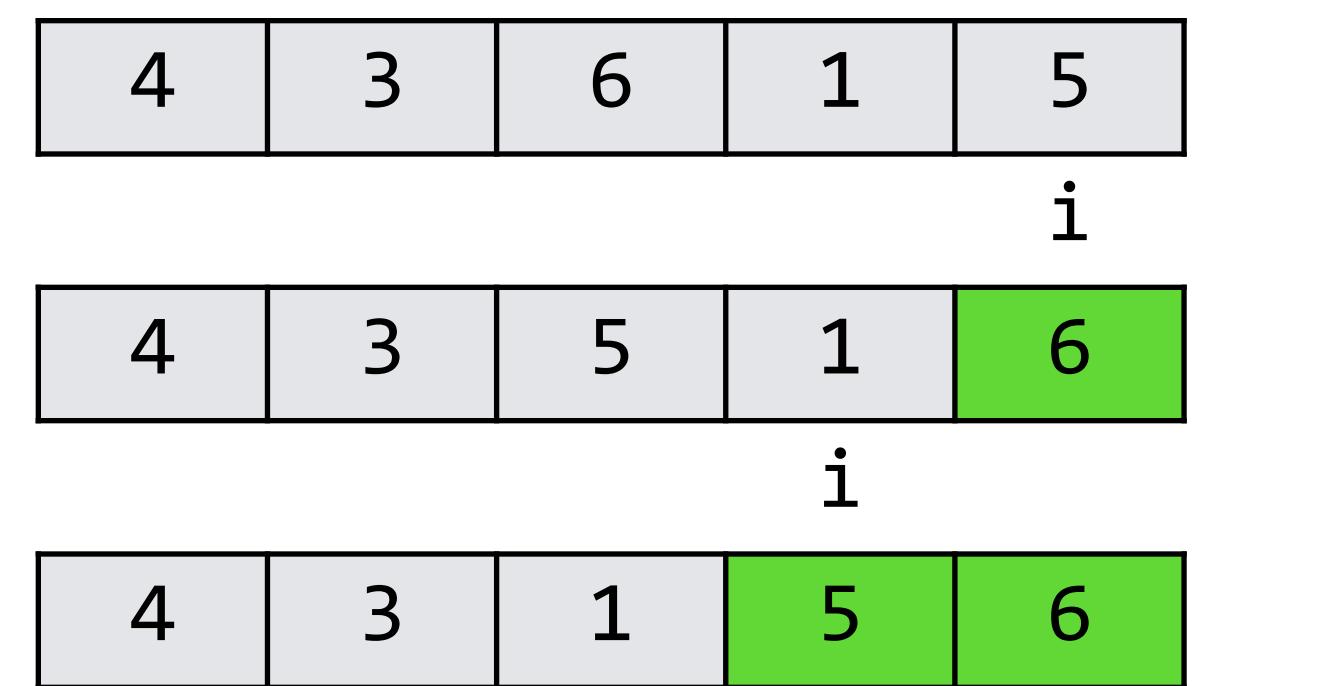
```
void ssort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        swap(a, i-1, max(a,i));  
}
```



Selection sort

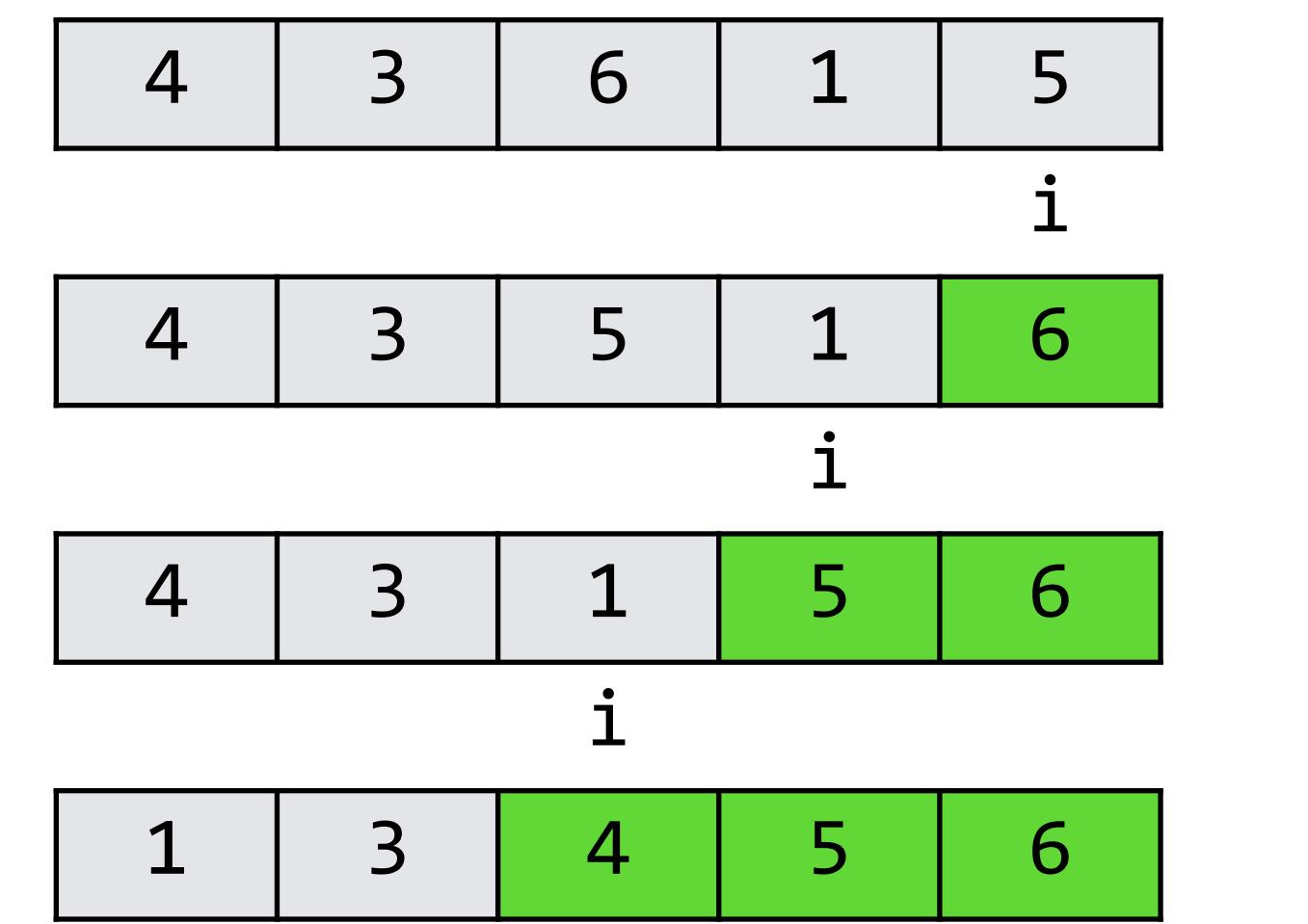
```
int max(int a[], int n) {  
    int m = 0, i;  
    for (i = 1; i < n; i++)  
        if (a[i] > a[m]) m = i;  
    return m;  
}
```

```
void ssort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        swap(a, i-1, max(a,i));  
}
```



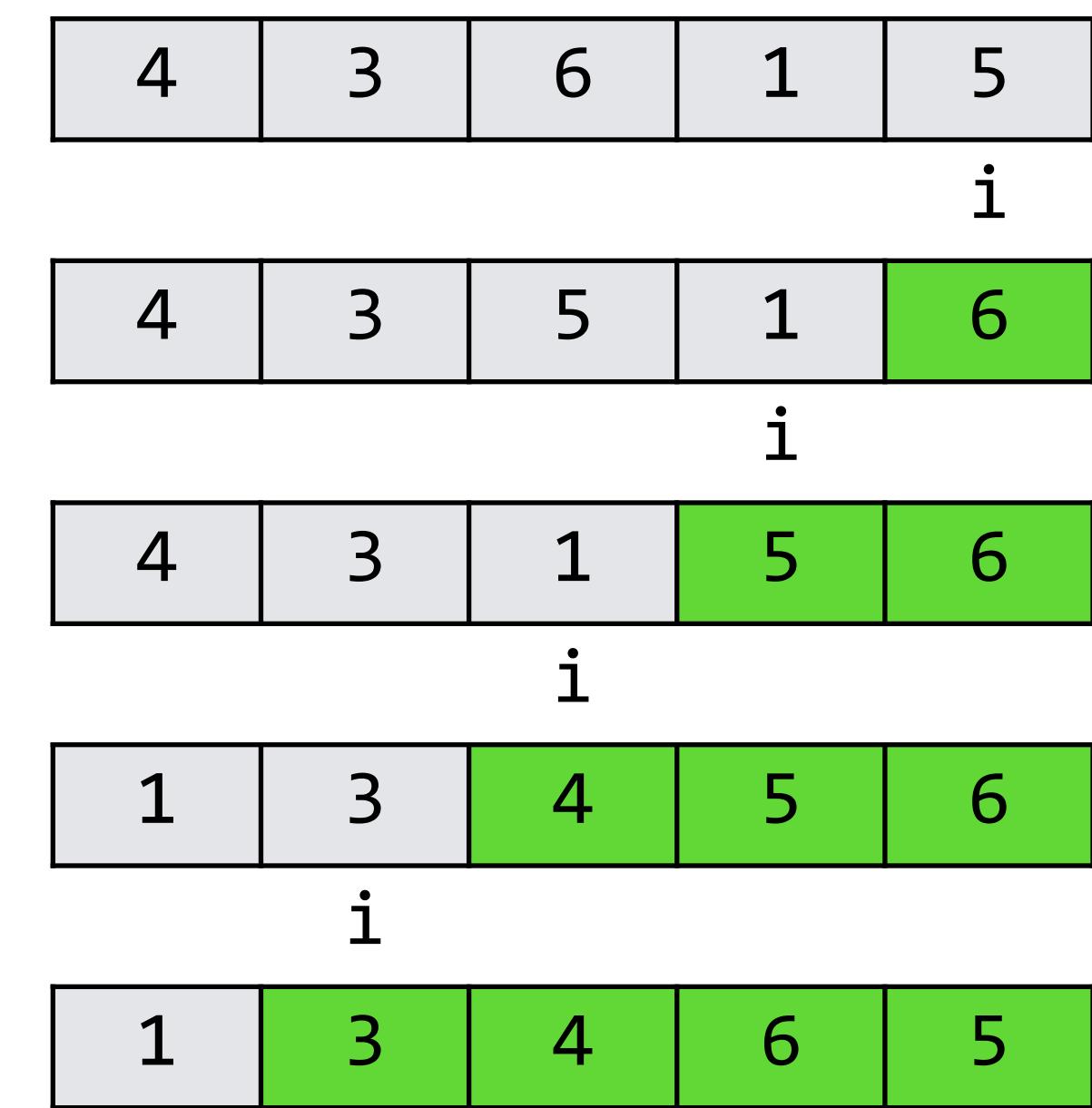
Selection sort

```
int max(int a[], int n) {  
    int m = 0, i;  
    for (i = 1; i < n; i++)  
        if (a[i] > a[m]) m = i;  
    return m;  
}  
  
void ssort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        swap(a, i-1, max(a,i));  
}
```



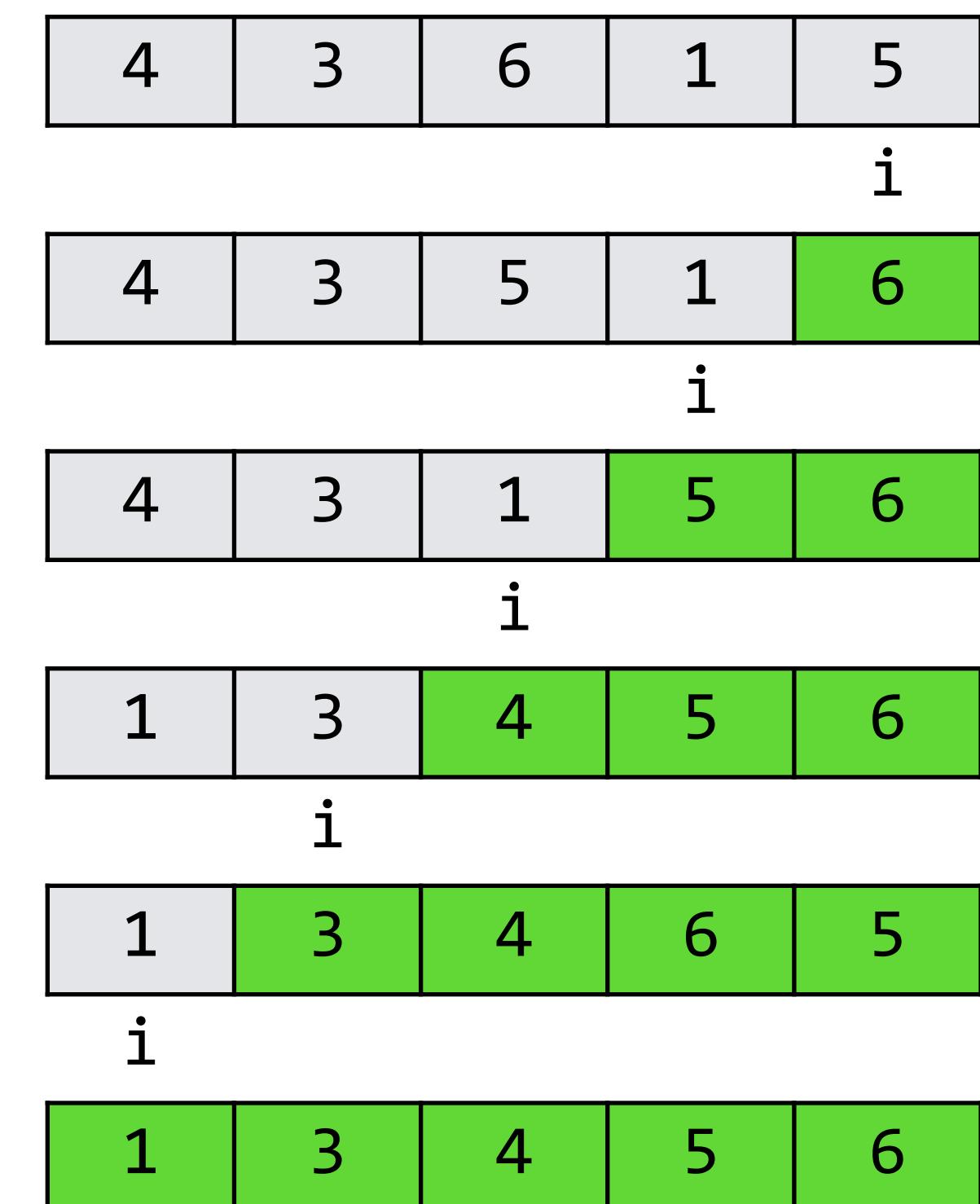
Selection sort

```
int max(int a[], int n) {  
    int m = 0, i;  
    for (i = 1; i < n; i++)  
        if (a[i] > a[m]) m = i;  
    return m;  
}  
  
void ssort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        swap(a, i-1, max(a,i));  
}
```



Selection sort

```
int max(int a[], int n) {  
    int m = 0, i;  
    for (i = 1; i < n; i++)  
        if (a[i] > a[m]) m = i;  
    return m;  
}  
  
void ssort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        swap(a, i-1, max(a,i));  
}
```



Selection sort

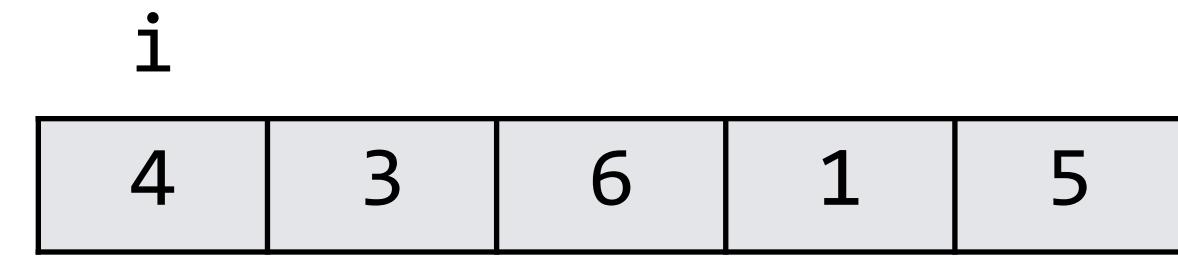
```
void ssort(int a[], int n) {  
    int i, j, m;  
    for (i = n; i > 0; i--) {  
        m = 0;  
        for (j = 0; j < i; j++)  
            if (a[j] > a[m]) m = j;  
        swap(a, i-1, m);  
    }  
}
```

Selection sort

```
void ssort(int a[], int n) {  
    int i, j, m;  
    for (i = 0; i < n; i++) {  
        m = i;  
        for (j = i; j < n; j++)  
            if (a[j] < a[m]) m = j;  
        swap(a, i, m);  
    }  
}
```

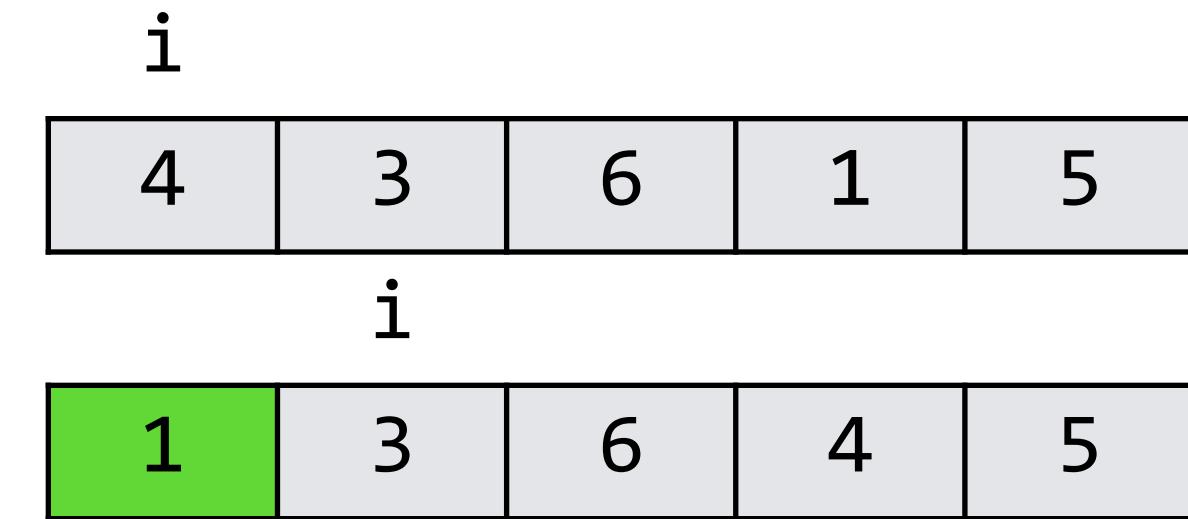
Selection sort

```
void ssort(int a[], int n) {  
    int i, j, m;  
    for (i = 0; i < n; i++) {  
        m = i;  
        for (j = i; j < n; j++)  
            if (a[j] < a[m]) m = j;  
        swap(a, i, m);  
    }  
}
```



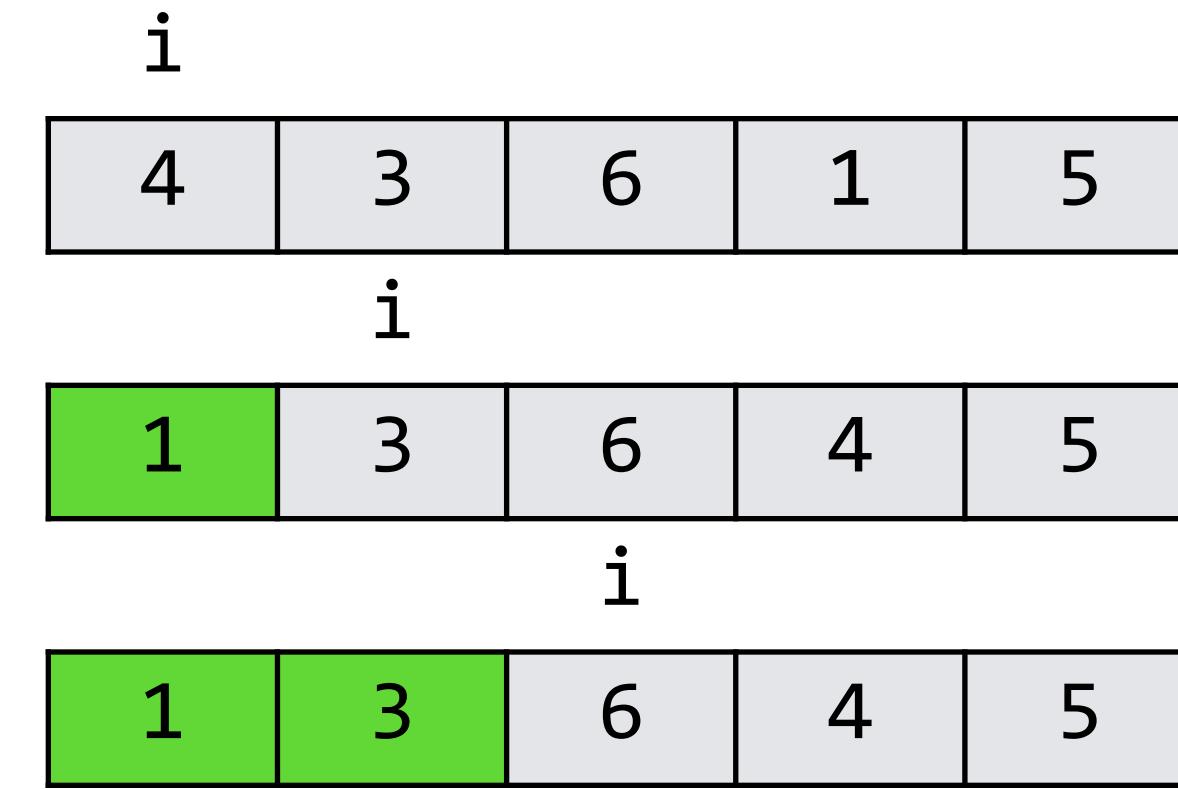
Selection sort

```
void ssort(int a[], int n) {  
    int i, j, m;  
    for (i = 0; i < n; i++) {  
        m = i;  
        for (j = i; j < n; j++)  
            if (a[j] < a[m]) m = j;  
        swap(a, i, m);  
    }  
}
```



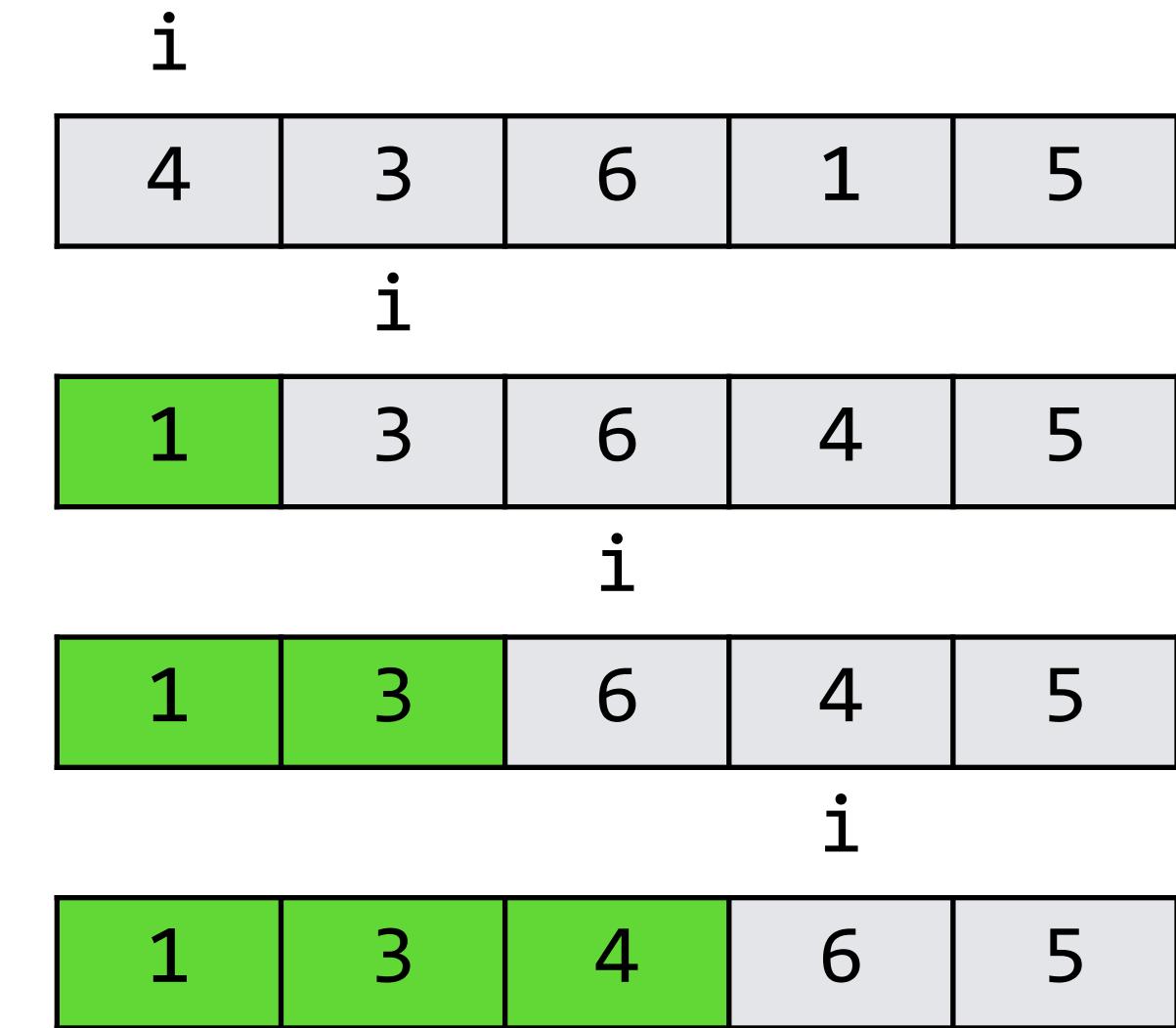
Selection sort

```
void ssort(int a[], int n) {  
    int i, j, m;  
    for (i = 0; i < n; i++) {  
        m = i;  
        for (j = i; j < n; j++)  
            if (a[j] < a[m]) m = j;  
        swap(a, i, m);  
    }  
}
```



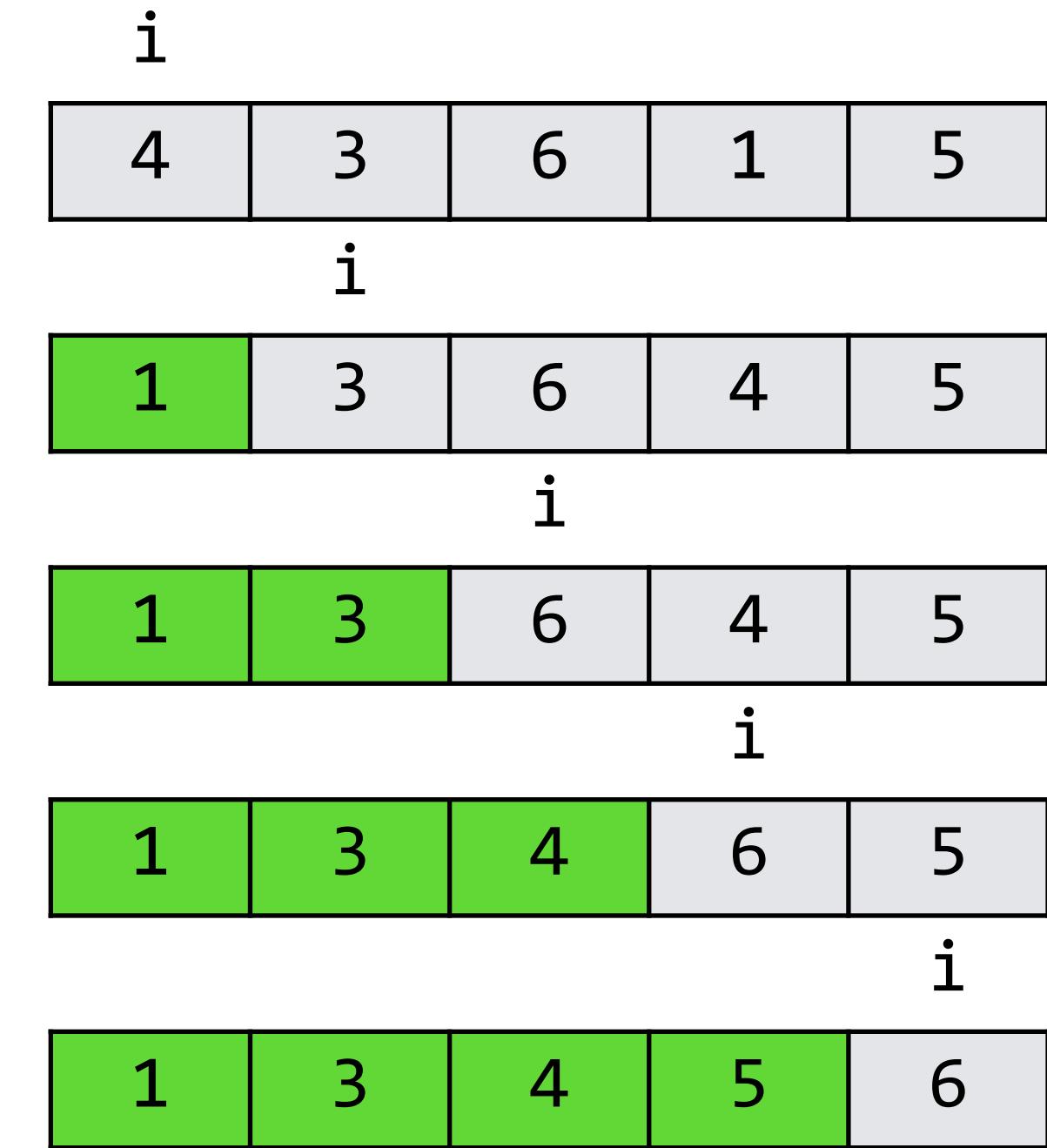
Selection sort

```
void ssort(int a[], int n) {  
    int i, j, m;  
    for (i = 0; i < n; i++) {  
        m = i;  
        for (j = i; j < n; j++)  
            if (a[j] < a[m]) m = j;  
        swap(a, i, m);  
    }  
}
```



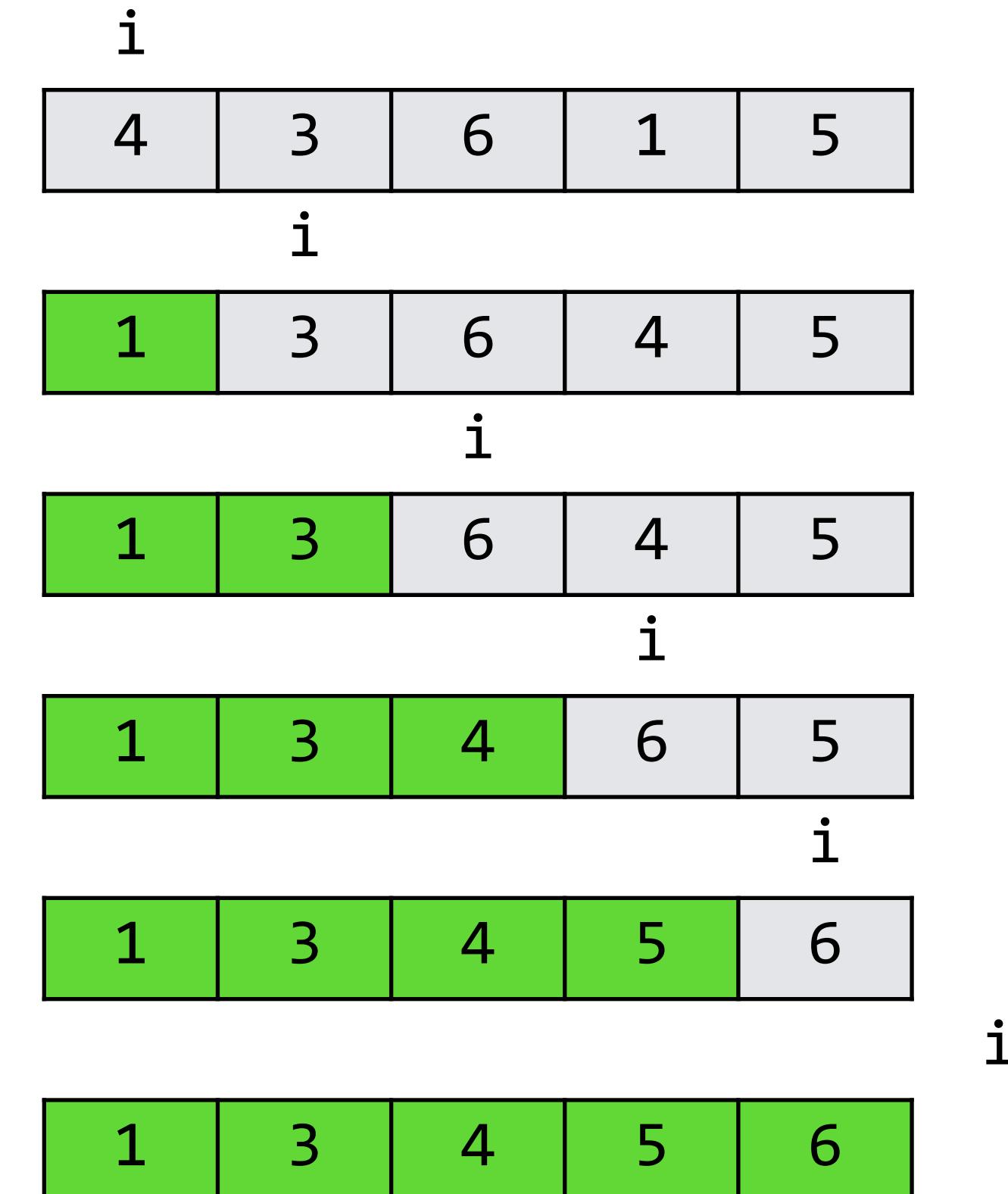
Selection sort

```
void ssort(int a[], int n) {  
    int i, j, m;  
    for (i = 0; i < n; i++) {  
        m = i;  
        for (j = i; j < n; j++)  
            if (a[j] < a[m]) m = j;  
        swap(a, i, m);  
    }  
}
```



Selection sort

```
void ssort(int a[], int n) {  
    int i, j, m;  
    for (i = 0; i < n; i++) {  
        m = i;  
        for (j = i; j < n; j++)  
            if (a[j] < a[m]) m = j;  
        swap(a, i, m);  
    }  
}
```



Bubble sort

```
void bubble(int a[], int n) {  
    int i;  
    for (i = 1; i < n; i++)  
        if (a[i-1] > a[i])  
            swap(a, i-1, i);  
}
```

```
void bsort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        bubble(a, i);  
}
```

Bubble sort

```
void bubble(int a[], int n) {  
    int i;  
    for (i = 1; i < n; i++)  
        if (a[i-1] > a[i])  
            swap(a, i-1, i);  
}
```

```
void bsort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        bubble(a, i);  
}
```

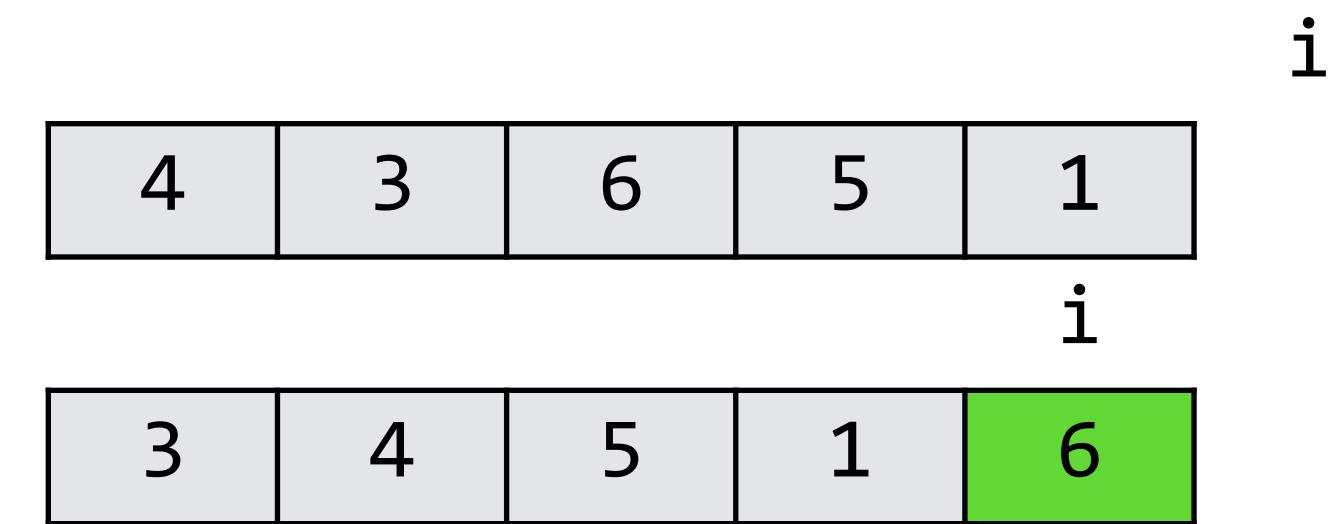
4	3	6	5	1
---	---	---	---	---

i

Bubble sort

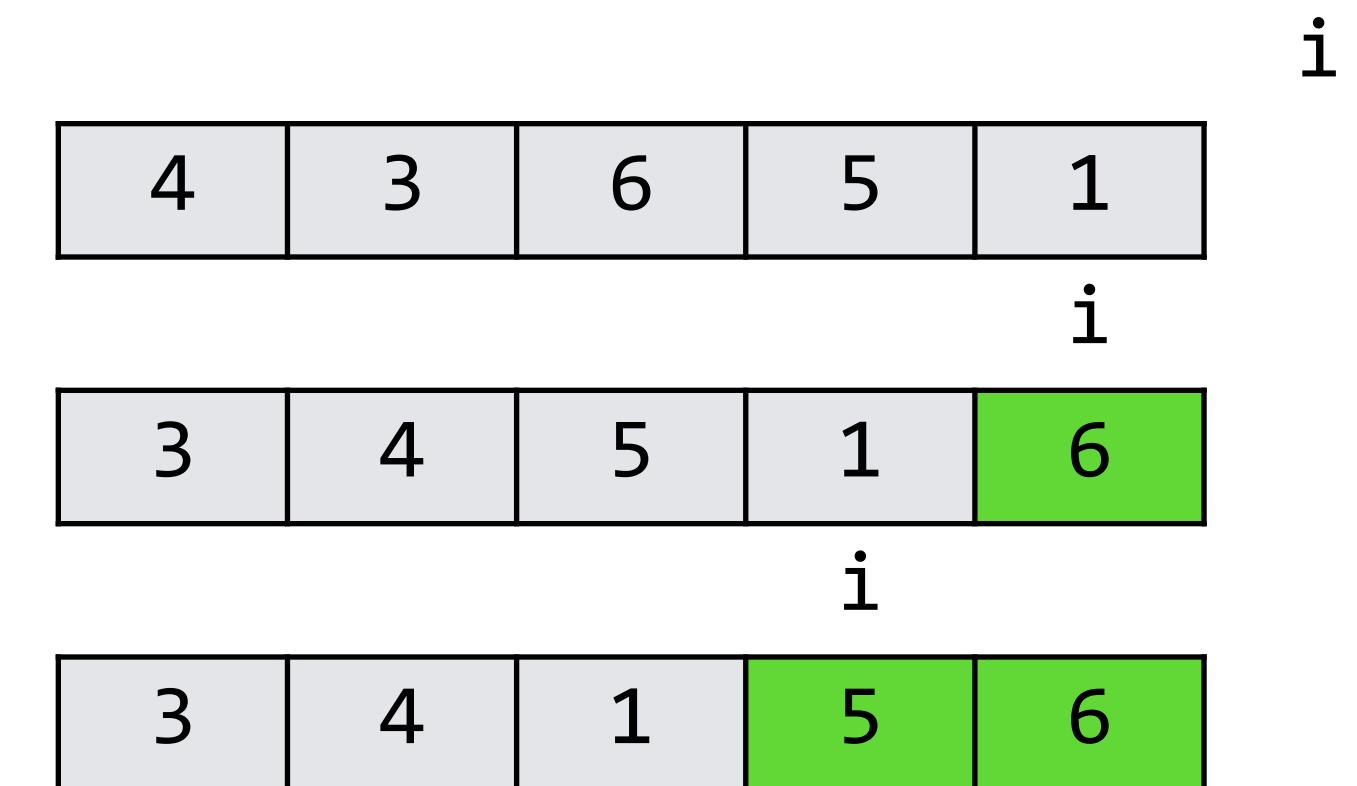
```
void bubble(int a[], int n) {  
    int i;  
    for (i = 1; i < n; i++)  
        if (a[i-1] > a[i])  
            swap(a, i-1, i);  
}
```

```
void bsort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        bubble(a, i);  
}
```



Bubble sort

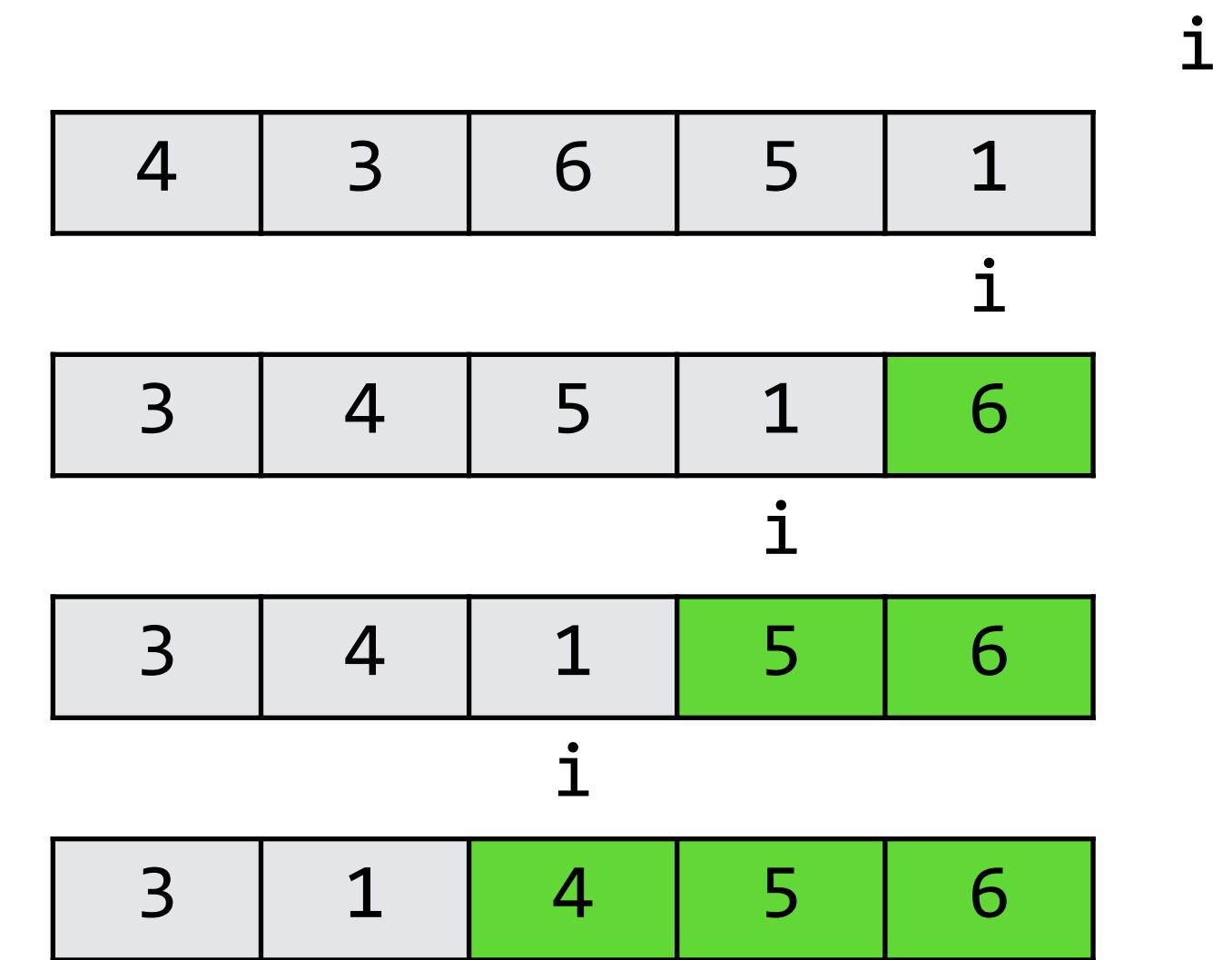
```
void bubble(int a[], int n) {  
    int i;  
    for (i = 1; i < n; i++)  
        if (a[i-1] > a[i])  
            swap(a, i-1, i);  
}
```



```
void bsort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        bubble(a, i);  
}
```

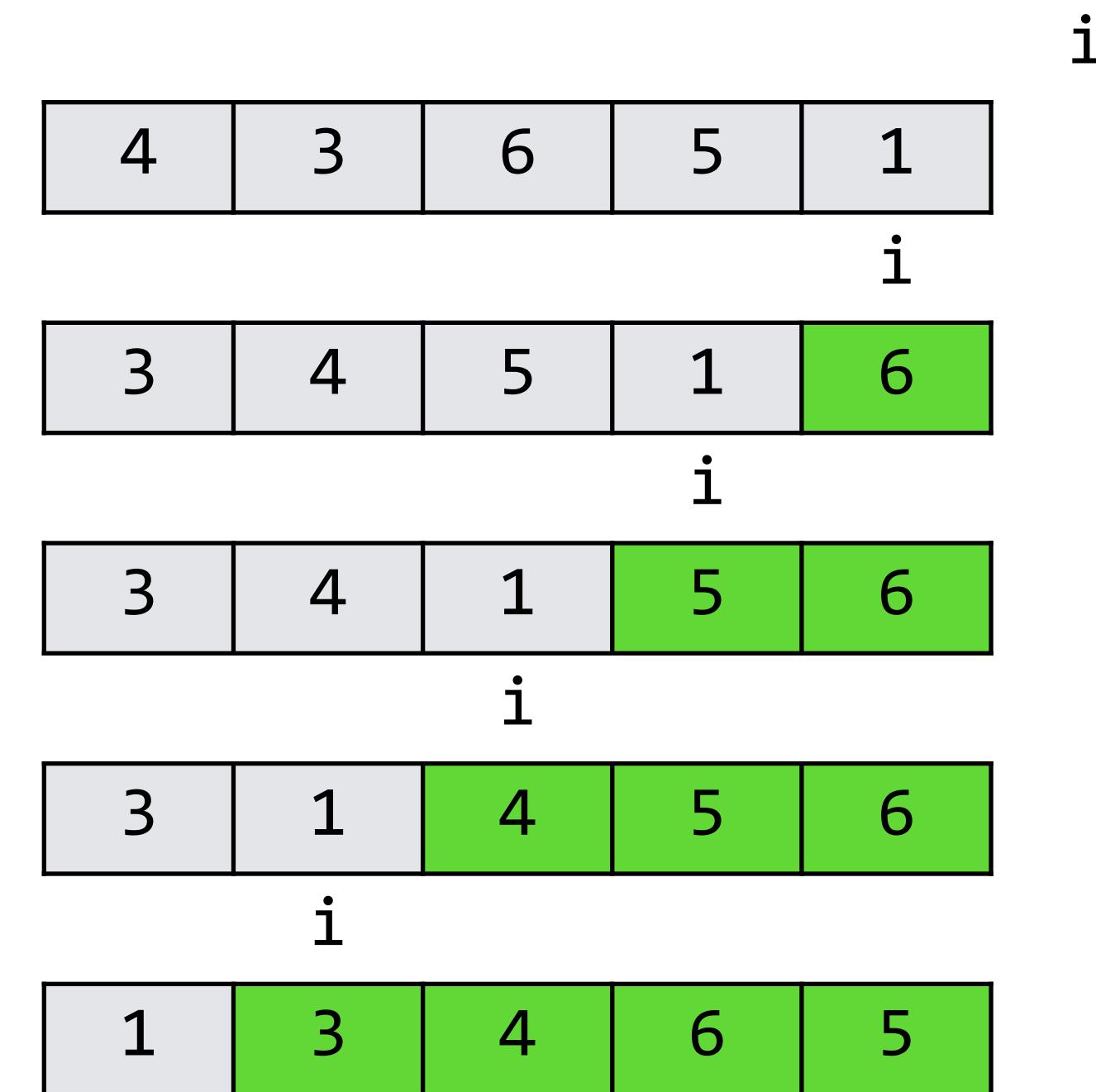
Bubble sort

```
void bubble(int a[], int n) {  
    int i;  
    for (i = 1; i < n; i++)  
        if (a[i-1] > a[i])  
            swap(a, i-1, i);  
}  
  
void bsort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        bubble(a, i);  
}
```



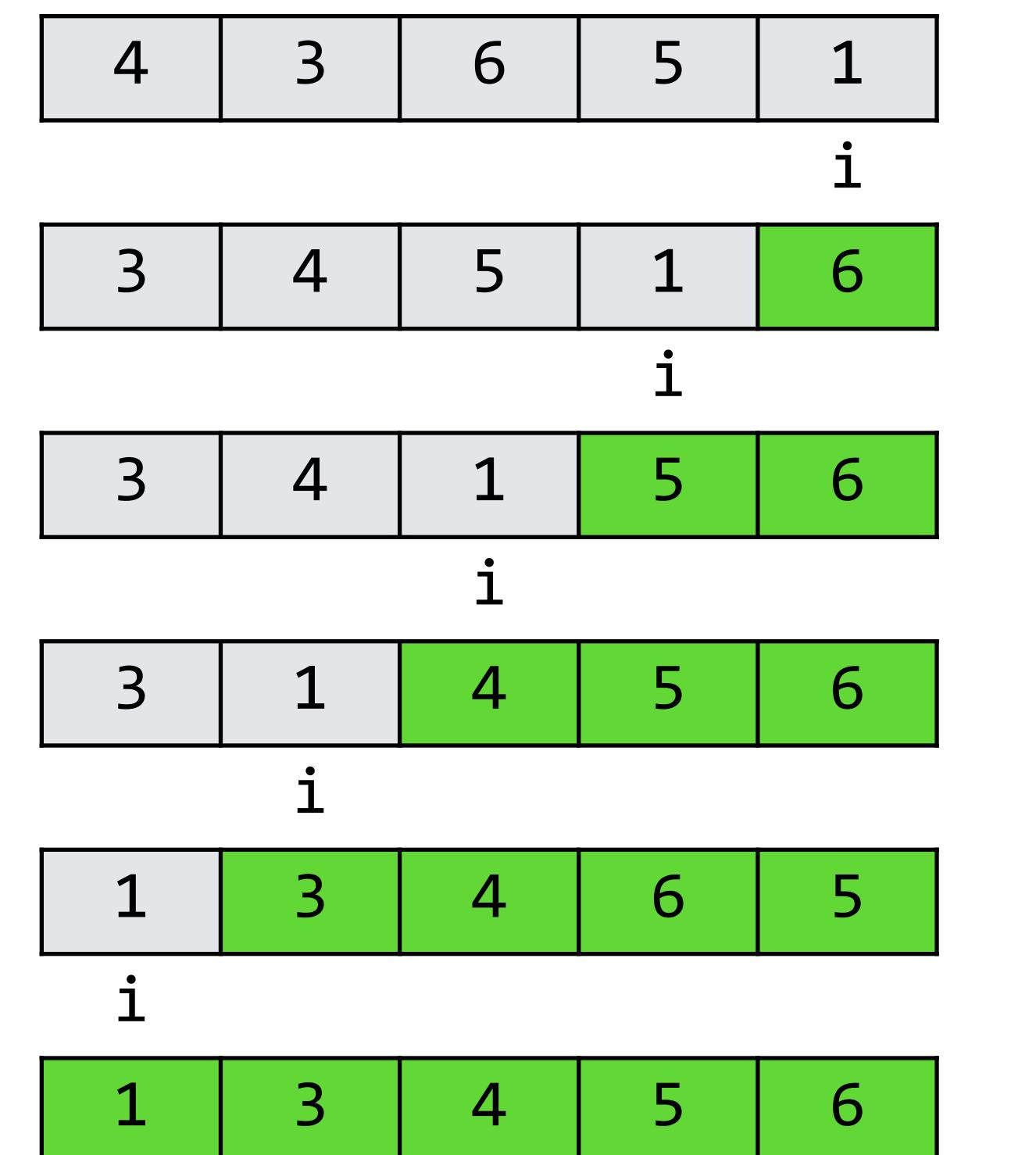
Bubble sort

```
void bubble(int a[], int n) {  
    int i;  
    for (i = 1; i < n; i++)  
        if (a[i-1] > a[i])  
            swap(a, i-1, i);  
}  
  
void bsort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        bubble(a, i);  
}
```



Bubble sort

```
void bubble(int a[], int n) {  
    int i;  
    for (i = 1; i < n; i++)  
        if (a[i-1] > a[i])  
            swap(a, i-1, i);  
}  
  
void bsort(int a[], int n) {  
    int i;  
    for (i = n; i > 0; i--)  
        bubble(a, i);  
}
```



Bubble sort

```
void bsort(int a[], int n) {  
    int i, j;  
    for (i = n; i > 0; i--)  
        for (j = 1; j < i; j++)  
            if (a[j-1] > a[j])  
                swap(a, j-1, j);  
}
```

Sinking sort

```
void bsort(int a[], int n) {  
    int i, j;  
    for (i = 0; i < n; i++)  
        for (j = n-1; j > i; j--)  
            if (a[j-1] > a[j])  
                swap(a, j-1, j);  
}
```

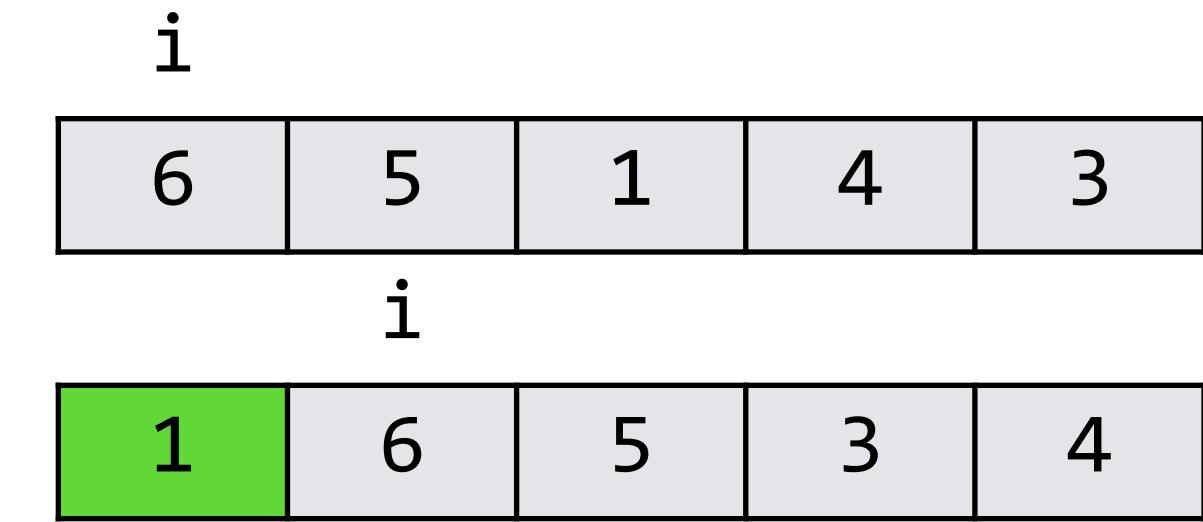
Sinking sort

```
void bsort(int a[], int n) {  
    int i, j;  
    for (i = 0; i < n; i++)  
        for (j = n-1; j > i; j--)  
            if (a[j-1] > a[j])  
                swap(a, j-1, j);  
}
```

i	6	5	1	4	3
---	---	---	---	---	---

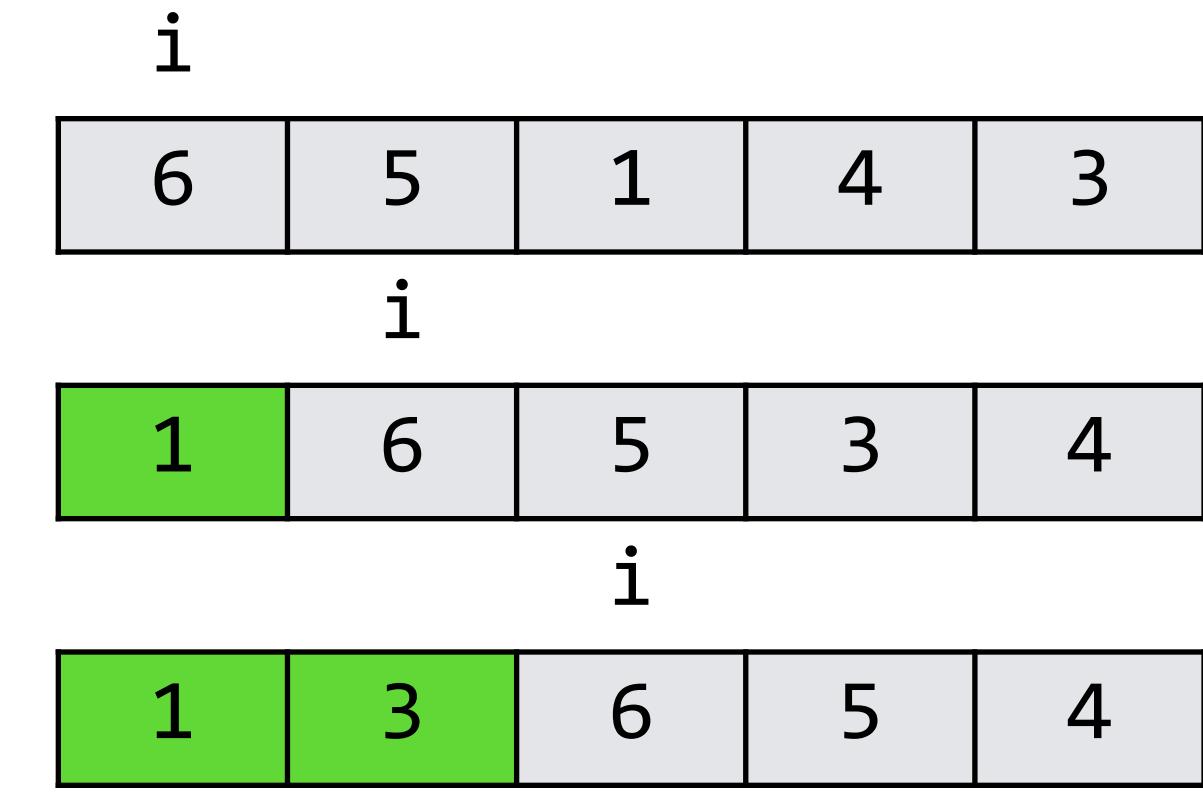
Sinking sort

```
void bsort(int a[], int n) {  
    int i, j;  
    for (i = 0; i < n; i++)  
        for (j = n-1; j > i; j--)  
            if (a[j-1] > a[j])  
                swap(a, j-1, j);  
}
```



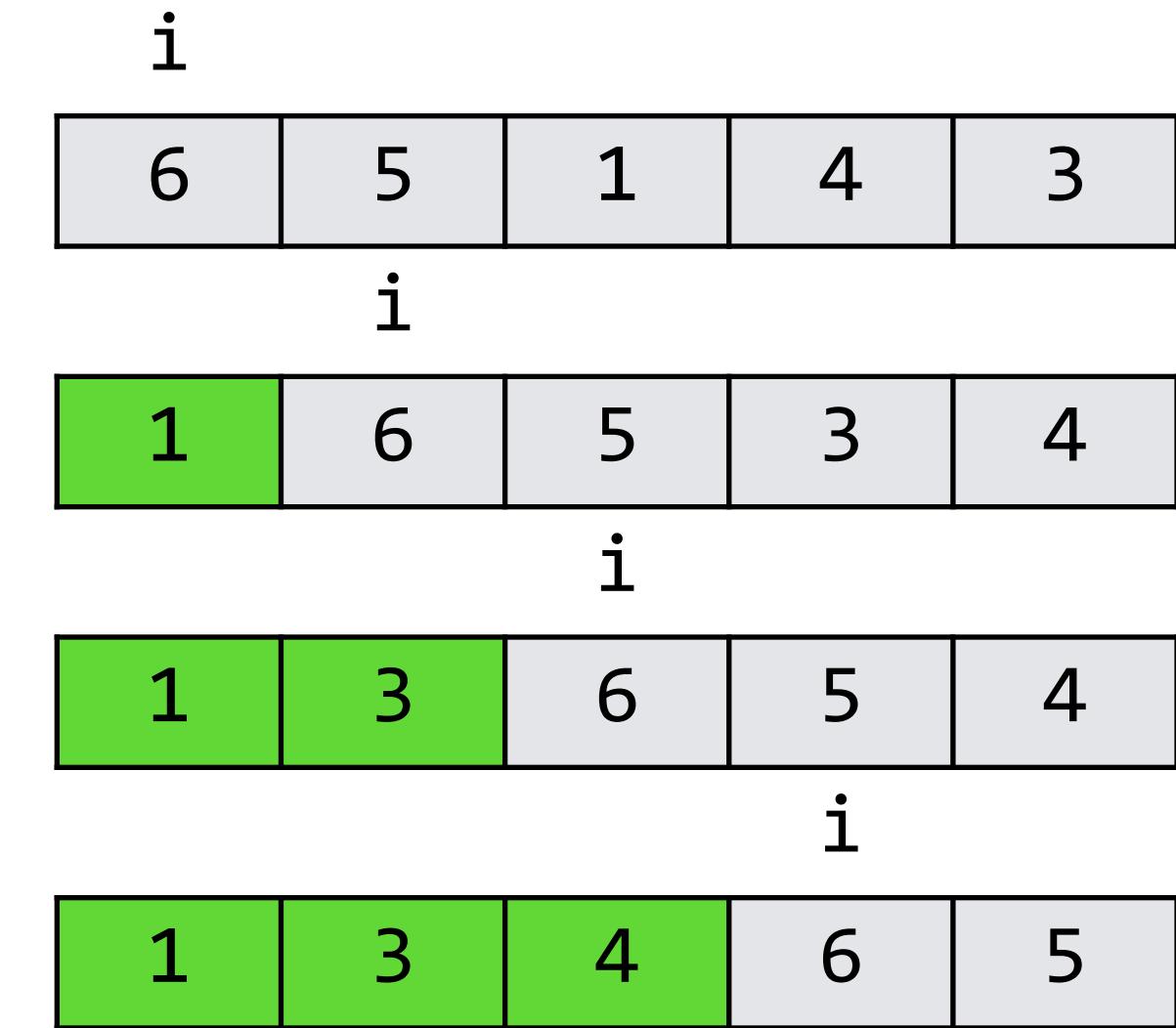
Sinking sort

```
void bsort(int a[], int n) {  
    int i, j;  
    for (i = 0; i < n; i++)  
        for (j = n-1; j > i; j--)  
            if (a[j-1] > a[j])  
                swap(a, j-1, j);  
}
```



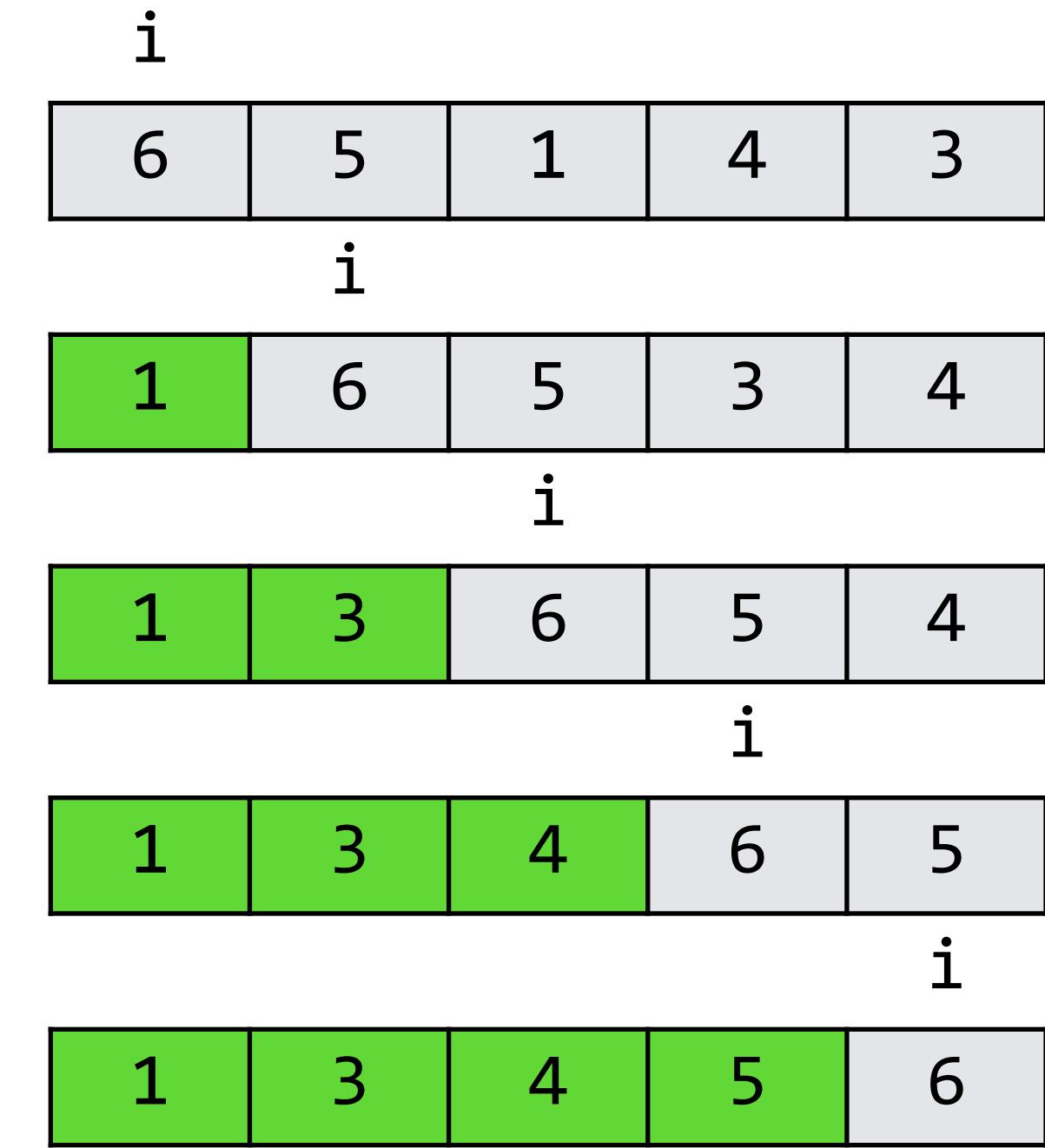
Sinking sort

```
void bsort(int a[], int n) {  
    int i, j;  
    for (i = 0; i < n; i++)  
        for (j = n-1; j > i; j--)  
            if (a[j-1] > a[j])  
                swap(a, j-1, j);  
}
```



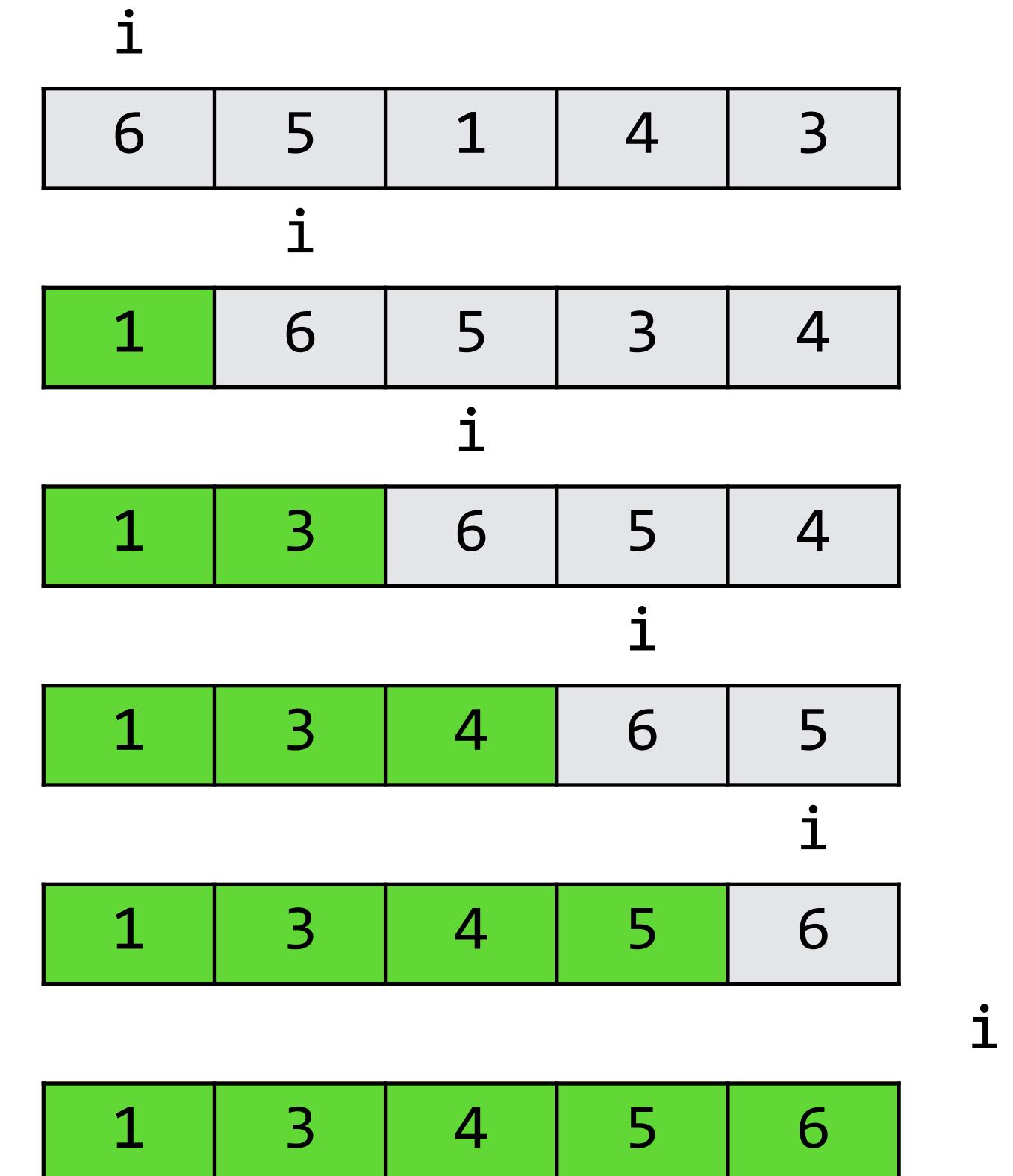
Sinking sort

```
void bsort(int a[], int n) {  
    int i, j;  
    for (i = 0; i < n; i++)  
        for (j = n-1; j > i; j--)  
            if (a[j-1] > a[j])  
                swap(a, j-1, j);  
}
```



Sinking sort

```
void bsort(int a[], int n) {  
    int i, j;  
    for (i = 0; i < n; i++)  
        for (j = n-1; j > i; j--)  
            if (a[j-1] > a[j])  
                swap(a, j-1, j);  
}
```



#15 Qual é mais rápido?

```
int v[1000];
for (int i = 0; i < 1000; i++) v[i] = i;
?sort(v, 1000);
```



#16 Qual faz menos escritas no array?

```
int v[1000];
for (int i = 0; i < 1000; i++) v[i] = 999-i;
?sort(v,1000);
```



Merge sort



John von Neumann, 1945

Merge sort

Merge sort

6	1	5	4	3
---	---	---	---	---

Merge sort

6	1	5	4	3
---	---	---	---	---

6	1	5	4	3
---	---	---	---	---

Merge sort

6	1	5	4	3
---	---	---	---	---

6	1
---	---

5	4	3
---	---	---

6

1

5 4 3

Merge sort

6	1	5	4	3
---	---	---	---	---

6	1
---	---

5	4	3
---	---	---

6

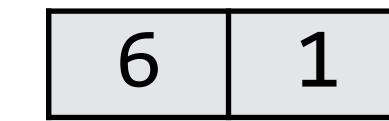
1

5	4	3
---	---	---

1	6
---	---

5	4	3
---	---	---

Merge sort



6

1

5	4	3
---	---	---

1	6
---	---

5	4	3
---	---	---

1	6
---	---

5

4	3
---	---

Merge sort



6

1



5

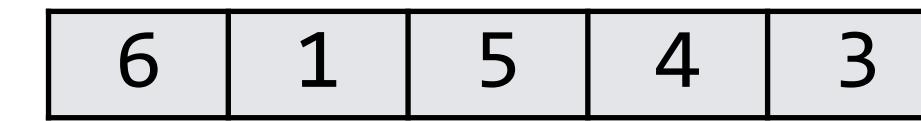


5

4

3

Merge sort



6

1



5



5

4

3

1

5

3

6

1

5

4

3

Merge sort



6

1

5	4	3
---	---	---

1	6
---	---

5	4	3
---	---	---

1	6
---	---

5

4	3
---	---

1	6
---	---

5

4	3
---	---

1	6
---	---

5

3	4
---	---

1	6
---	---

3	4	5
---	---	---

Merge sort



6

1

5	4	3
---	---	---

1	6
---	---

5	4	3
---	---	---

1	6
---	---

5

4	3
---	---

1	6
---	---

5

4	3
---	---

1	6
---	---

5

3	4
---	---

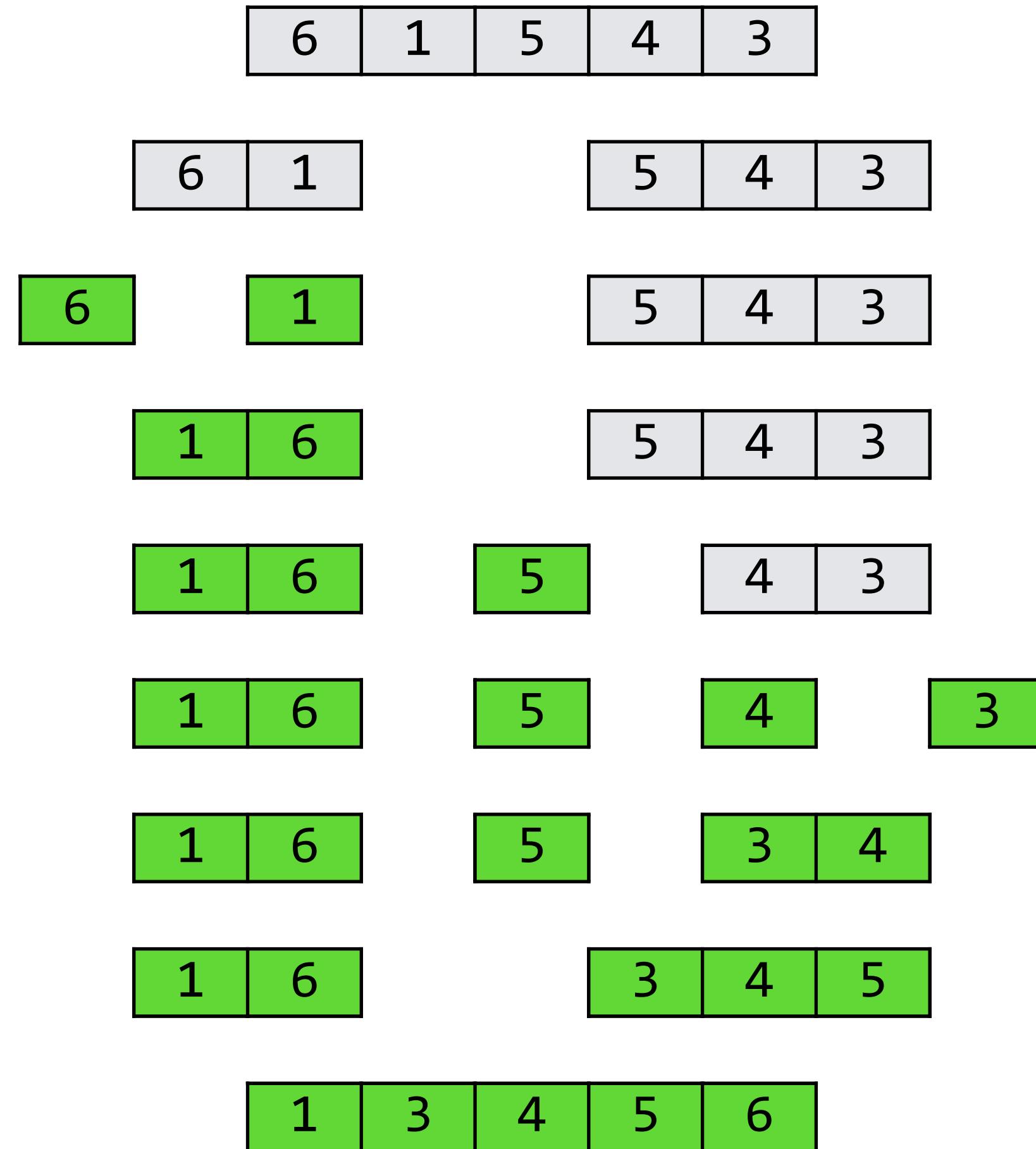
1	6
---	---

3	4	5
---	---	---

1	3	4	5	6
---	---	---	---	---

Merge sort

```
void msort(int a[], int n) {  
    if (n < 2) return;  
    int m = n/2;  
    int aux[n];  
    msort(a, m);  
    msort(a+m, n-m);  
    merge(a, m, a+m, n-m, aux);  
    copy(aux, a, n);  
}
```



Merge sort

```
void copy(int a[], int b[], int n) {  
    for (int i = 0; i < n; i++) b[i] = a[i];  
}  
  
void merge(int a[], int na, int b[], int nb, int r[]) {  
    ...  
}
```

Aula 9

Quick sort



Tony Hoare, 1959

Quick sort

Quick sort

6	1	5	4	3
---	---	---	---	---

Quick sort

6	1	5	4	3
---	---	---	---	---

1

3

6 5 4

Quick sort

6	1	5	4	3
---	---	---	---	---

1

3

6	5	4
---	---	---

1

3

6	5	4
---	---	---

Quick sort



Quick sort



Quick sort



Quick sort



Quick sort



Quick sort

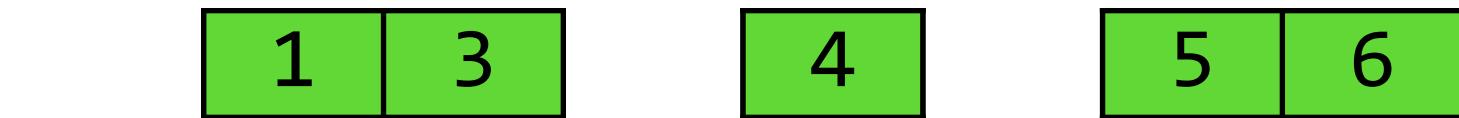


Quick sort



Quick sort

```
void qsort(int a[], int n) {  
    if (n < 2) return;  
    int p = partition(a, n-1, a[n-1]);  
    swap(a, p, n-1);  
    qsort(a, p);  
    qsort(a+p+1, n-p-1);  
}
```



Quick sort

```
int partition(int a[], int n, int x) {  
    ...  
}
```

Eficiência

Algoritmo	Melhor	Pior
Insertion sort	N	N^2
Selection sort	N^2	N^2
Bubble sort	N	N^2
Merge sort	$N \cdot \log_2 N$	$N \cdot \log_2 N$
Quick sort	$N \cdot \log_2 N$	N^2



Counting sort

```
void csort(int a[], int n) {  
    int c[10] = {0};  
    int i, j, k;  
    for (i = 0; i < n; i++) c[a[i]]++;  
    i = 0;  
    for (j = 0; j < 10; j++)  
        for (k = 0; k < c[j]; k++)  
            a[i++] = j;  
}
```

Counting sort

```
void csort(int a[], int n) {  
    int c[10] = {0};  
    int i, j, k;  
    for (i = 0; i < n; i++) c[a[i]]++;  
    i = 0;  
    for (j = 0; j < 10; j++)  
        for (k = 0; k < c[j]; k++)  
            a[i++] = j;  
}
```

4	3	9	4	1
---	---	---	---	---

0	1	0	1	2	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---

Counting sort

```
void csort(int a[], int n) {  
    int c[10] = {0};  
    int i, j, k;  
    for (i = 0; i < n; i++) c[a[i]]++;  
    i = 0;  
    for (j = 0; j < 10; j++)  
        for (k = 0; k < c[j]; k++)  
            a[i++] = j;  
}
```

1	3	4	4	9
---	---	---	---	---

0	1	0	1	2	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---

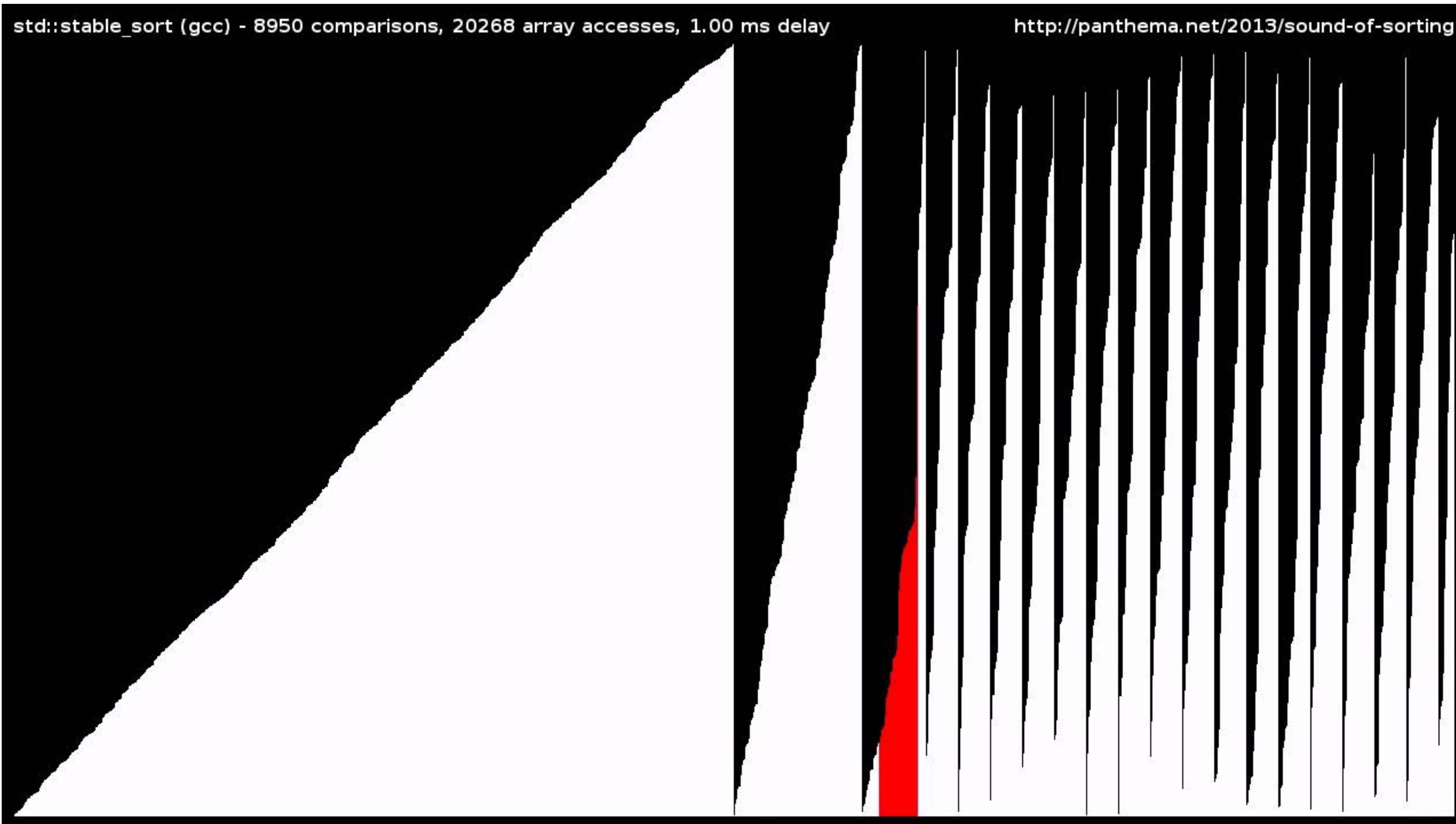
Let's dance



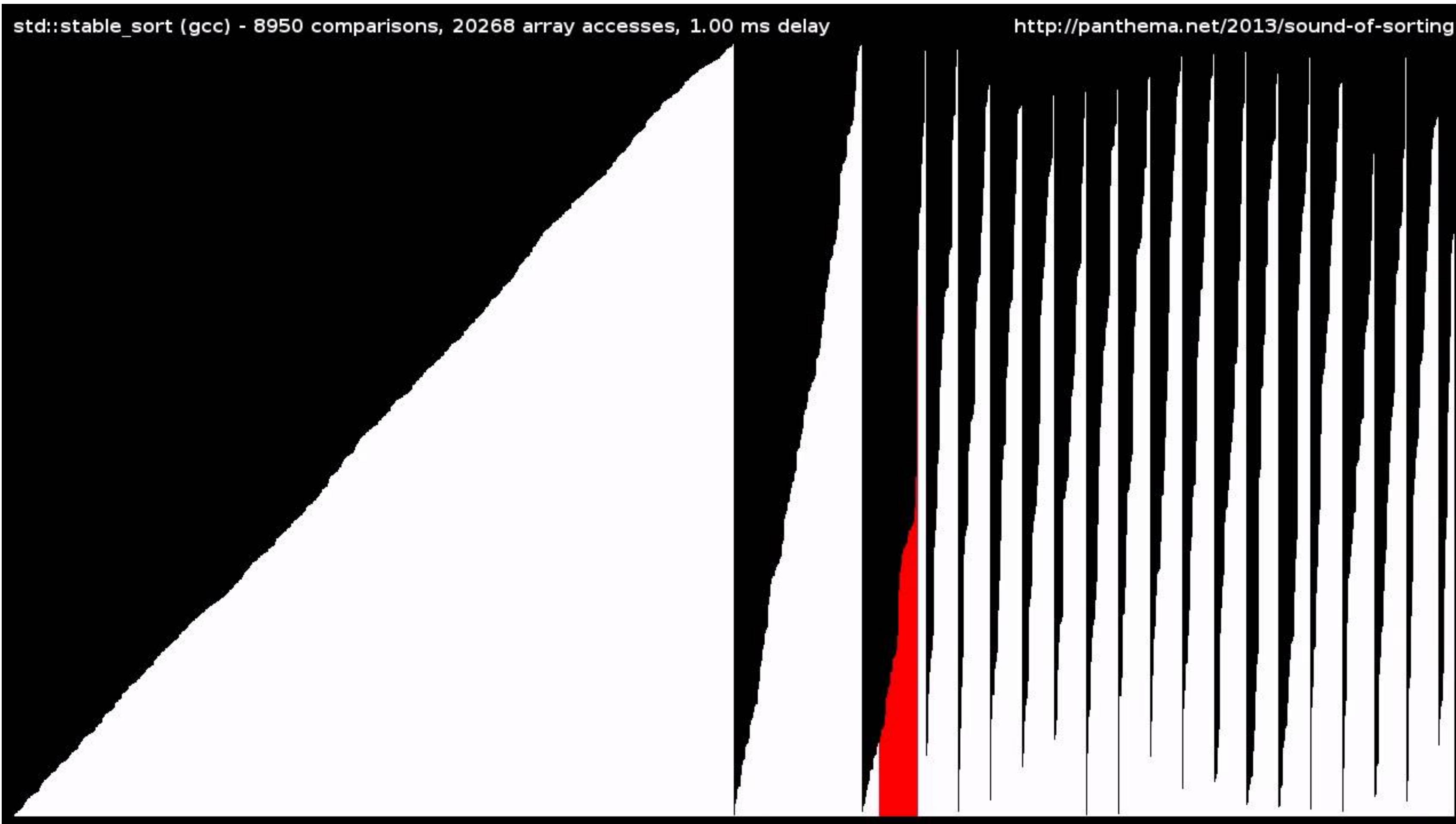
Let's dance



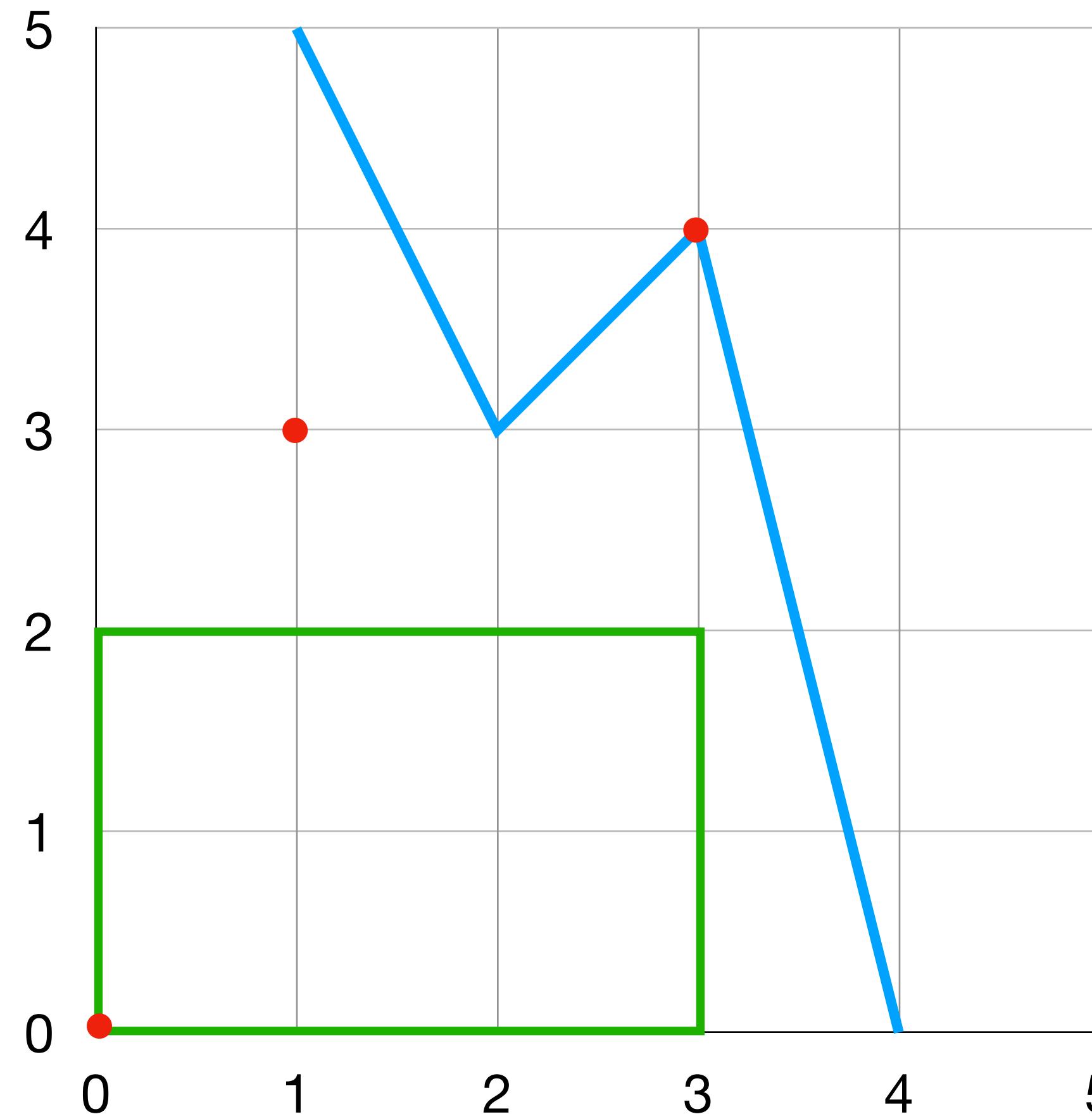
The sound of sorting



The sound of sorting



Pontos, rectângulos e poligonais



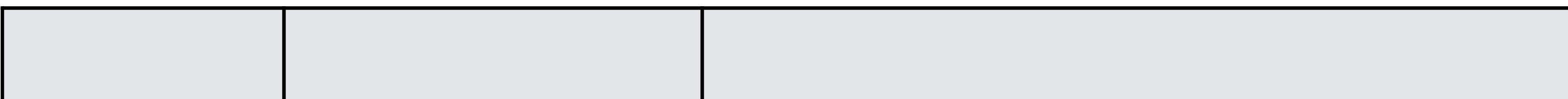
Declaração de structs

```
struct {  
    tipo1 id1;  
    ...  
    tipon idn;  
} var;
```

var.id₁

...

var.id_n



Inicialização de structs

```
struct {  
    tipo1 id1;  
    ...  
    tipon idn;  
} var = {expr1, ..., exprn};
```

var.id₁

...

var.id_n

<i>expr₁</i>			<i>expr_n</i>
-------------------------	--	--	-------------------------

Um ponto

```
struct {  
    float x;  
    float y;  
} o = {0.0, 0.0};
```

o.x	o.y
0.0	0.0

Declaração de um novo tipo struct

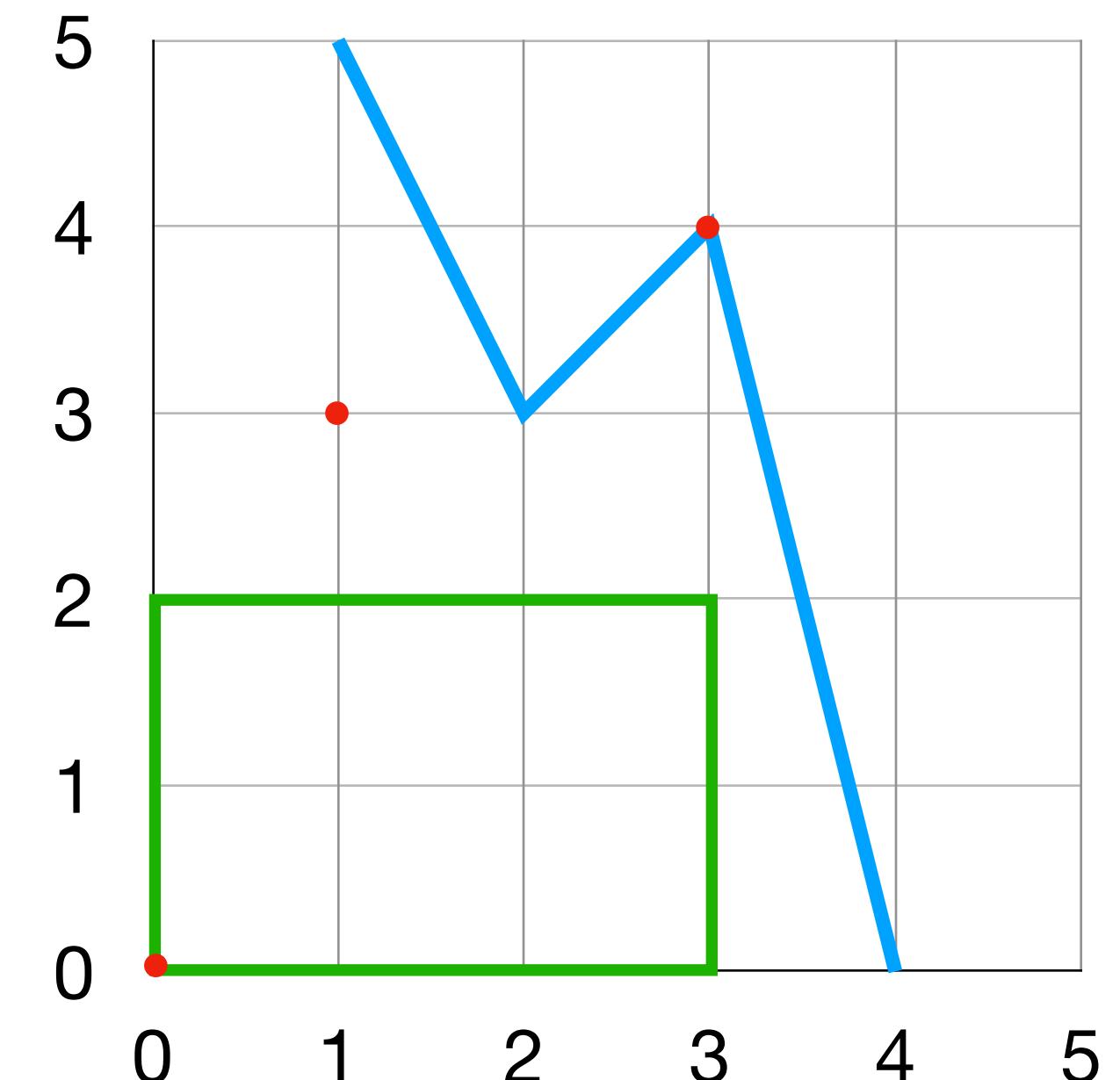
```
struct id {  
    tipo1 id1;  
    ...  
    tipon idn;  
};  
  
struct id var;
```

Pontos, rectângulos e poligonais

```
struct ponto {  
    float x;  
    float y;  
};  
  
struct rectangulo {  
    struct ponto pt1, pt2;  
};  
  
#define MAX 100  
  
struct poligonal {  
    int tamanho;  
    struct ponto pontos[MAX];  
};
```

Pontos, rectângulos e poligonais

```
struct ponto o = {0,0};  
struct ponto a = {1,3};  
struct ponto b = {3,4};  
struct rectangulo r = {o, {3,2}};  
struct poligonal l = {4, {{1,5}, {2,3}, b, {4,0}}};
```

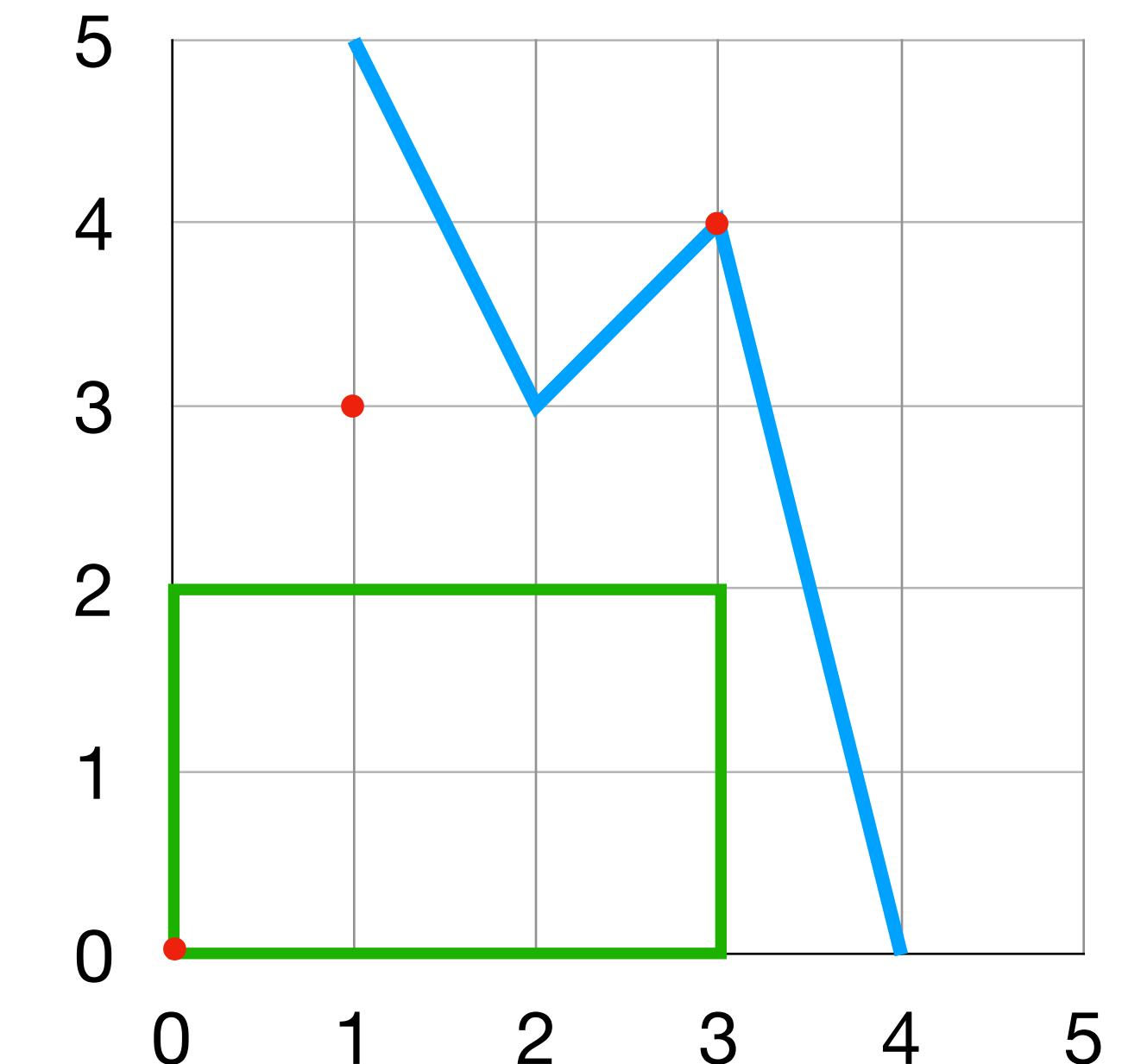


Operações com structs

- Pode-se aceder aos membros
- Podem-se copiar na totalidade
 - Em particular, são copiadas quando passadas a ou devolvidas por funções
- Não se podem comparar directamente
 - Tem que se comparar os membros

Pontos, rectângulos e poligonais

```
struct ponto o = {0,0};  
struct ponto a = {1,3};  
struct ponto b;  
b.x = 3;  
b.y = 4;  
struct rectangulo r;  
r.pt1 = o;  
r.pt2.x = b.x;  
r.pt2.y = 2;  
struct poligonal l = {4, {{1,5}, {2,3}}};  
l.pontos[2] = b;  
l.pontos[3].x = 4;  
l.pontos[3].y = 0;
```



Sinónimos de tipos

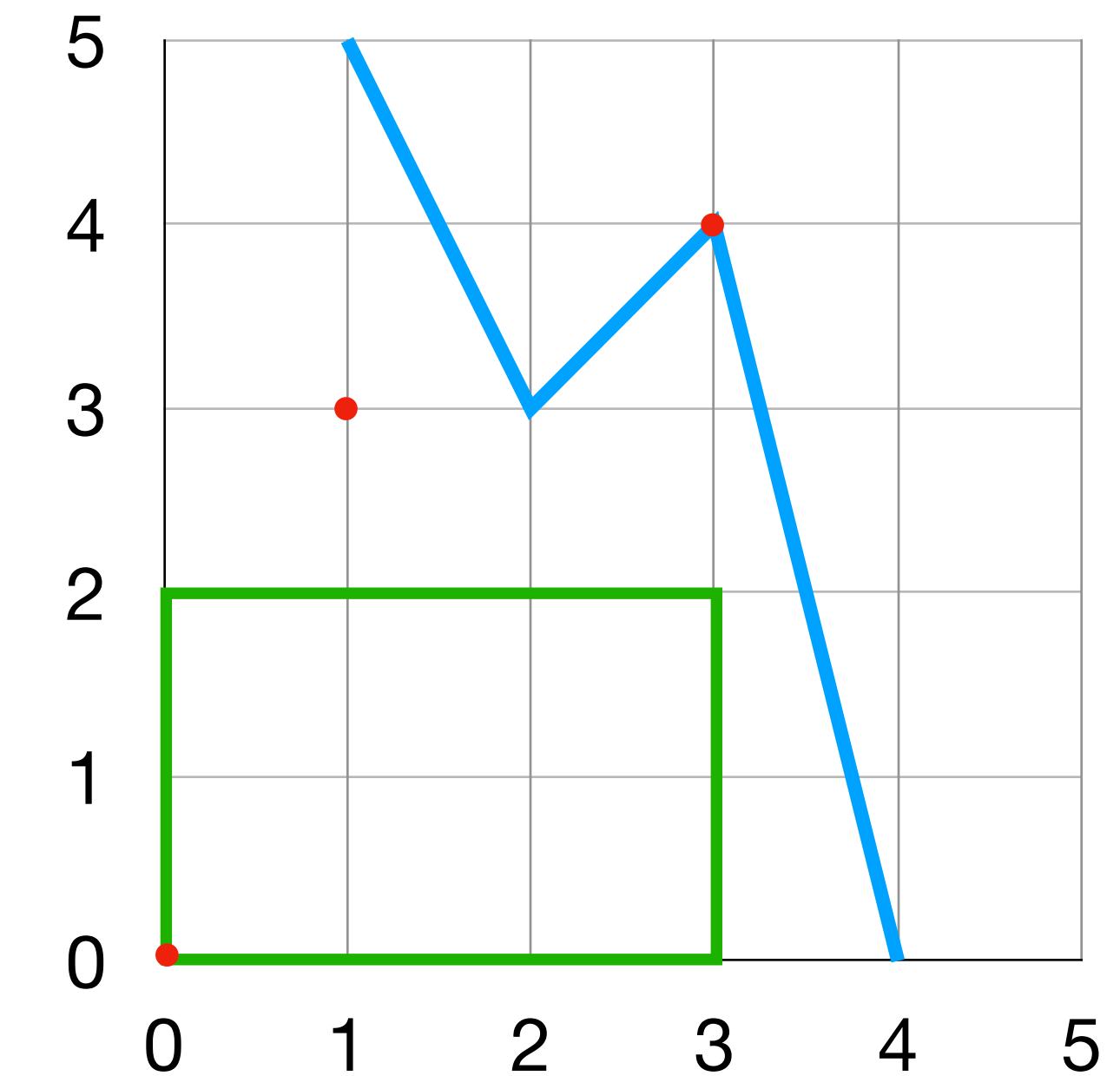
```
typedef tipo id;
```

Pontos, rectângulos e poligonais

```
struct pt {  
    float x;  
    float y;  
};  
typedef struct pt ponto;  
  
typedef struct {  
    ponto pt1, pt2;  
} rectangulo;  
  
#define MAX 100  
  
struct poligonal {  
    int tamanho;  
    ponto pontos[MAX];  
};  
typedef struct poligonal poligonal;
```

Pontos, rectângulos e poligonais

```
ponto o = {0,0};  
ponto a = {1,3};  
ponto b;  
b.x = 3;  
b.y = 4;  
rectangulo r;  
r.pt1 = o;  
r.pt2.x = b.x;  
r.pt2.y = 2;  
poligonal l = {4, {{1,5}, {2,3}}};  
l.pontos[2] = b;  
l.pontos[3].x = 4;  
l.pontos[3].y = 0;
```



#17 Qual o primeiro ponto de m no final?

```
ponto o = {0,0};  
poligonal l = {4, {{1,5}, {2,3}, {3,4}, {4,0}}};  
poligonal m = l;  
l.pontos[0] = o;  
printf("%.1f,.1f}\n", m.pontos[0].x, m.pontos[0].y);
```



Distância entre dois pontos

```
#include <math.h>

float dist(ponto a, ponto b) {
    return sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2));
}

int main() {
    ponto o = {0,0}, a = {1,3};
    printf("%.2f\n",dist(o,a));
    return 0;
}
```

Área de um rectângulo

```
#include <math.h>

float area(rectangulo r) {
    return fabs(r.pt1.x - r.pt2.x) * fabs(r.pt1.y - r.pt2.y);
}

int main() {
    rectangulo r = {{0,0}, {3,2}};
    printf("%.2f\n",area(r));
    return 0;
}
```

#18 Qual a área de q?

```
rectangulo figura(ponto p, float d) {  
    rectangulo r = {p, {p.x + d, p.y + d}};  
    return r;  
}  
  
int main() {  
    ponto a = {1,3};  
    rectangulo q = figura(a, 2);  
    printf("%.1f\n", area(q));  
    return 0;  
}
```



Aula 10

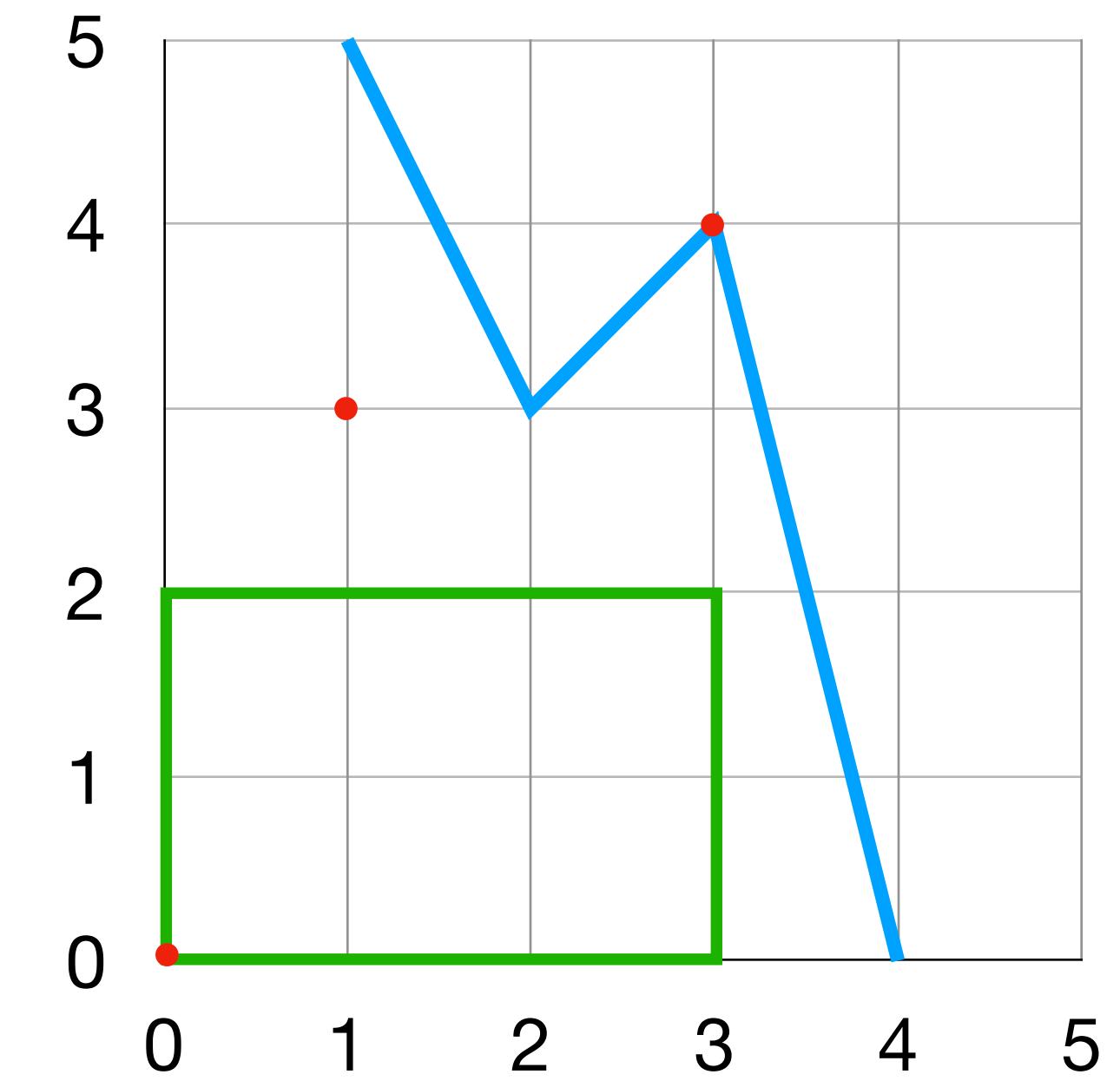
Poligonais coloridas

```
#define RED 0
#define GREEN 1
#define BLUE 2

#define MAX 100

struct poligonal {
    int tamanho;
    ponto pontos[MAX];
    int cor;
};

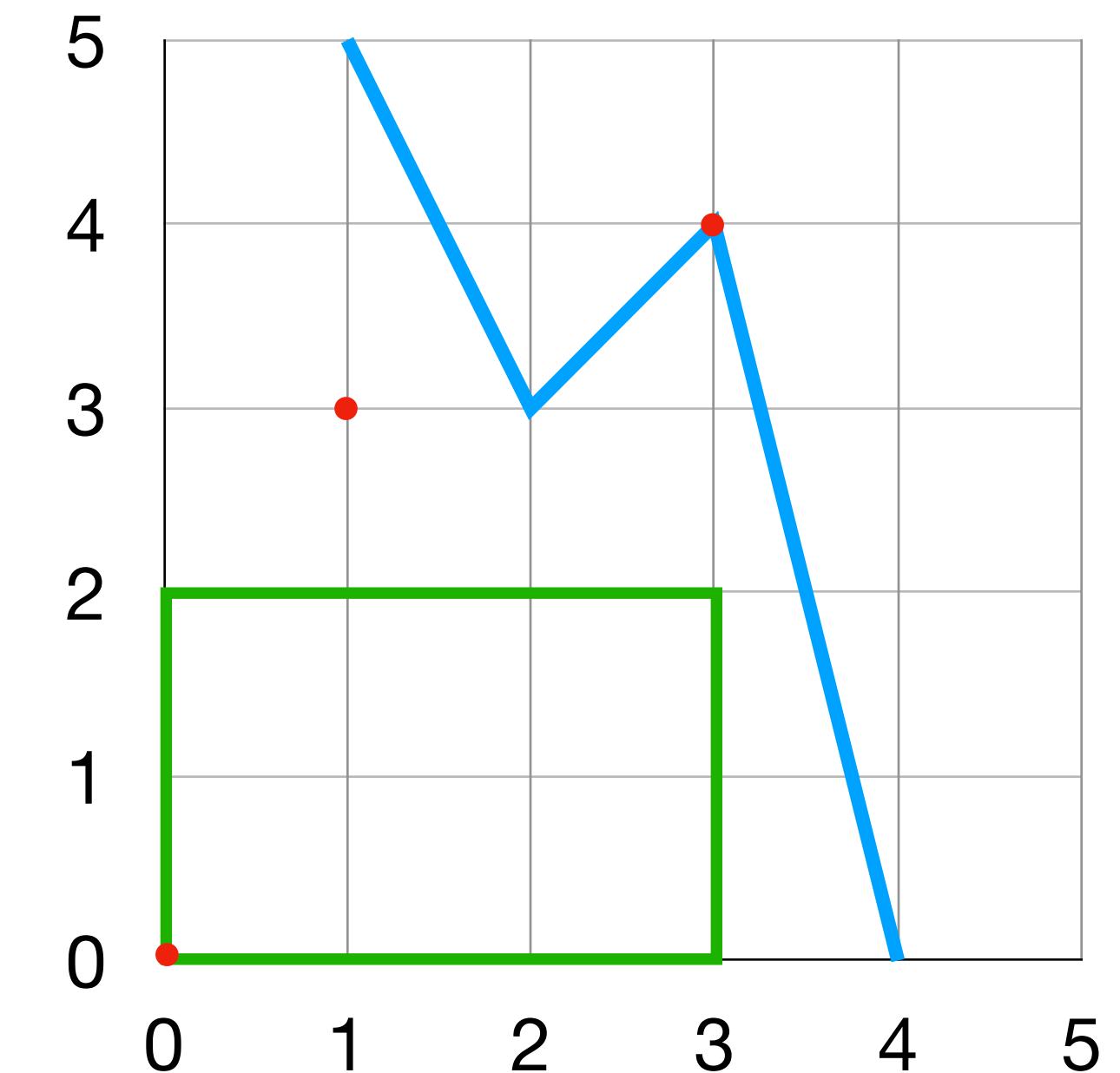
typedef struct poligonal poligonal;
```



Poligonais coloridas

```
ponto a = {1,5};  
ponto b = {2,3};  
ponto c = {3,4};  
ponto d = {4,0};
```

```
poligonal l = {4, {a,b,c,d}, BLUE};
```



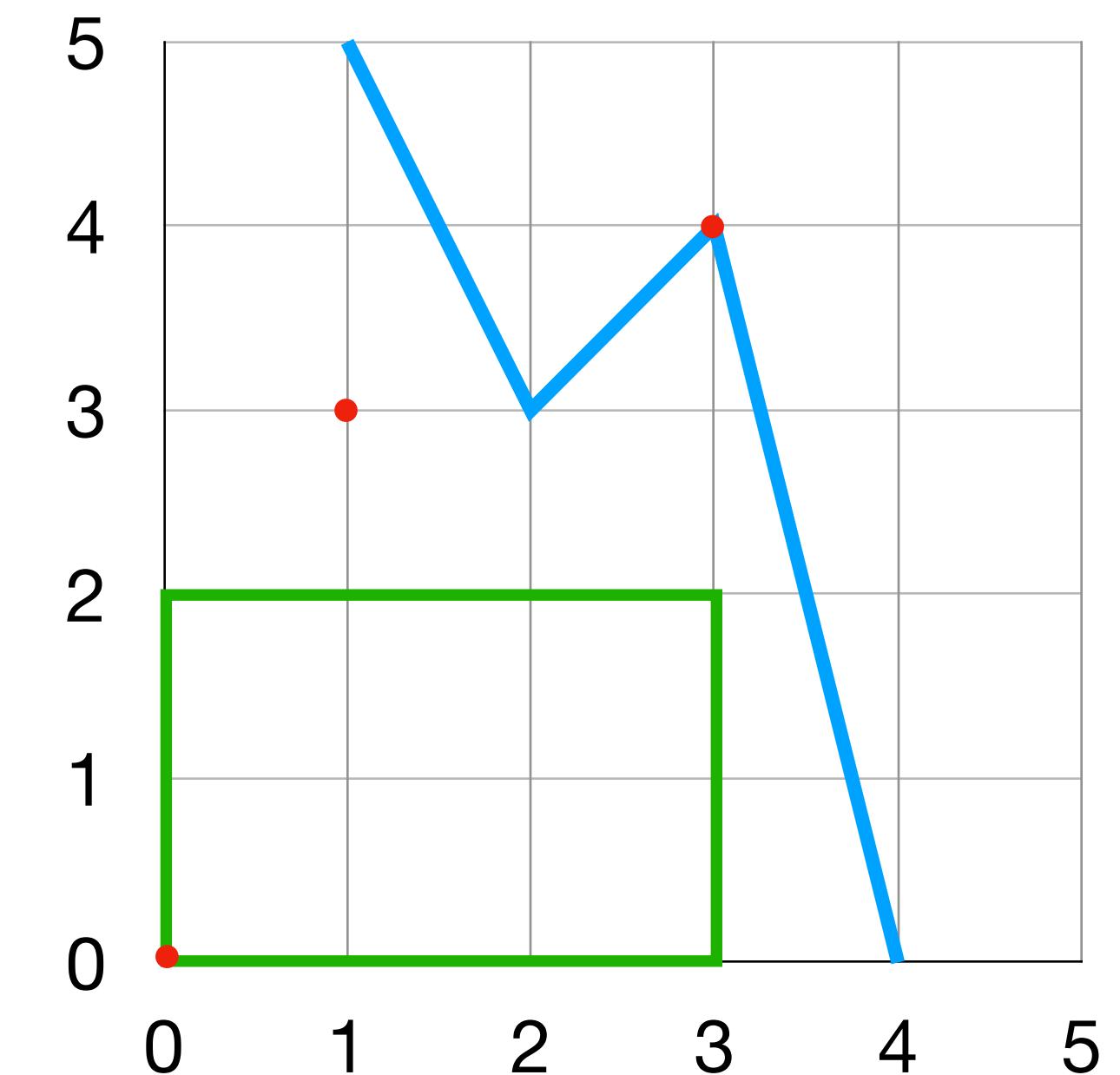
Declaração de um novo tipo enum

```
enum id {  
    id1,  
    ...  
    idn  
};
```

```
enum id var;
```

Poligonais coloridas

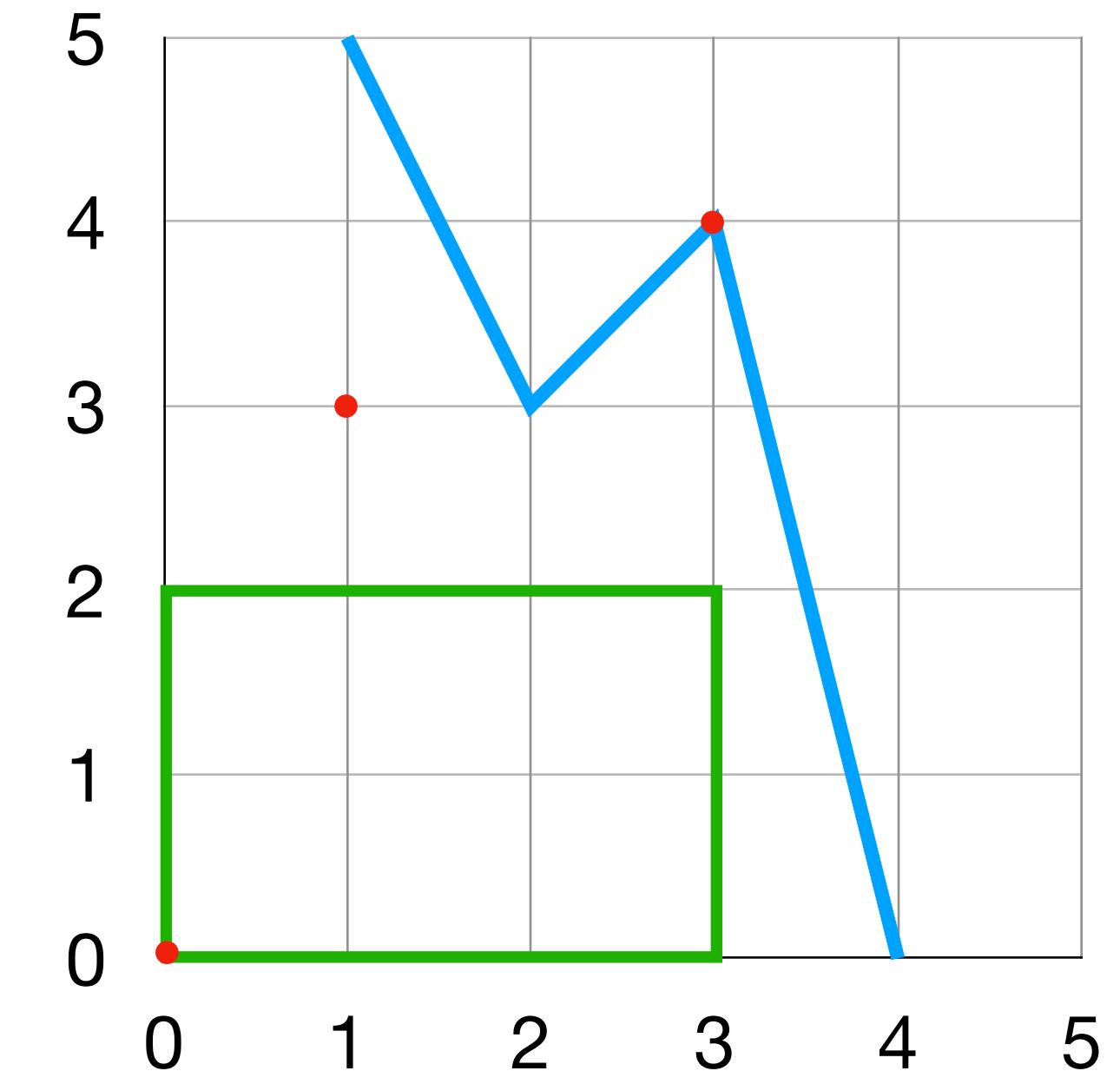
```
enum cor {RED, GREEN, BLUE};  
typedef enum cor cor;  
  
#define MAX 100  
  
struct poligonal {  
    int tamanho;  
    ponto pontos[MAX];  
    cor cor;  
};  
typedef struct poligonal poligonal;
```



Poligonais coloridas

```
ponto a = {1,5};  
ponto b = {2,3};  
ponto c = {3,4};  
ponto d = {4,0};
```

```
poligonal l = {4, {a,b,c,d}, BLUE};
```



Comprimento de uma poligonal

```
float comprimento(poligonal l) {
    float r = 0;
    for (int i = 1; i < l.tamanho; i++)
        r += dist(l.pontos[i-1], l.pontos[i]);
    return r;
}

int main() {
    poligonal l = {4, {{1,5}, {2,3}, {3,4}, {4,0}}, BLUE};
    printf("%.1f\n", comprimento(l));
    return 0;
}
```



Comprimento de uma poligonal

```
float comprimento(poligonal *l) {
    float r = 0;
    for (int i = 1; i < (*l).tamanho; i++)
        r += dist((*l).pontos[i-1], (*l).pontos[i]);
    return r;
}

int main() {
    poligonal l = {4, {{1,5}, {2,3}, {3,4}, {4,0}}, BLUE};
    printf("%.1f\n", comprimento(&l));
    return 0;
}
```

Acesso a membro via apontador

$(\ast var).id$

\equiv

$var->id$

Comprimento de uma poligonal

```
float comprimento(poligonal *l) {
    float r = 0;
    for (int i = 1; i < l->tamanho; i++)
        r += dist(l->pontos[i-1], l->pontos[i]);
    return r;
}

int main() {
    poligonal l = {4, {{1,5}, {2,3}, {3,4}, {4,0}}, BLUE};
    printf("%.1f\n", comprimento(&l));
    return 0;
}
```

Estender uma poligonal

```
int estende(poligonal *l, ponto p) {
    if (l->tamanho == MAX) return 1;
    l->pontos[l->tamanho] = p;
    l->tamanho++;
    return 0;
}

int main() {
    ponto o = {0,0};
    poligonal l = {4, {{1,5}, {2,3}, {3,4}, {4,0}}, BLUE};
    estende(&l, o);
    printf("%.1f\n", comprimento(&l));
    return 0;
}
```



Arrays dinâmicos

- Arrays que crescem ou diminuem conforme a necessidade de espaço
- Suportados nativamente em muitas linguagens modernas
 - Java (`ArrayList`), C++ (`std::vector`), Python (`list`), ...
- Tipicamente, quando ficam cheios duplicam a capacidade
 - Operação pouco eficiente, mas executada poucas vezes
- C não tem suporte nativo para arrays dinâmicos
 - Mas podem ser implementados com operações explícitas de gestão de memória

malloc e free

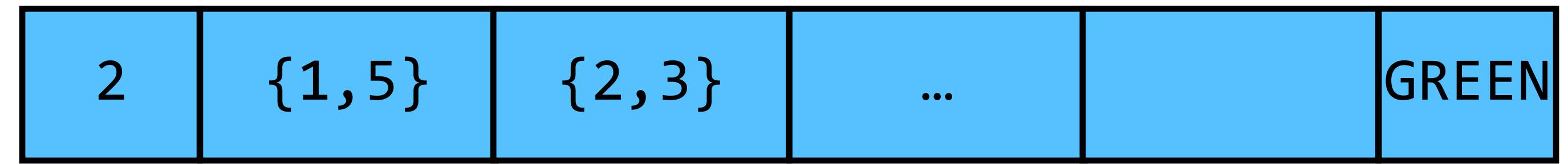
- A função `malloc` aloca (reserva) um dado número de bytes na *heap*
 - Devolve um apontador para o primeiro byte alocado
 - Os bytes alocados são contíguos
 - Devolve o apontador `NULL` se não foi possível alocar
 - Deve ser usada a primitiva `sizeof` para calcular o número de bytes a alocar
- A função `free` liberta espaço previamente alocado
- Declaradas na biblioteca `<stdlib.h>`

Poligonal com array na stack

```
#define MAX 100

typedef struct {
    int tamanho;
    ponto pontos[MAX];
    cor cor;
} poligonal;

int main() {
    ponto a = {1,5}, b = {2,3};
    poligonal l;
    l.tamanho = 0;
    l.cor = GREEN;
    estende(&l,a);
    estende(&l,b);
    return 0;
}
```

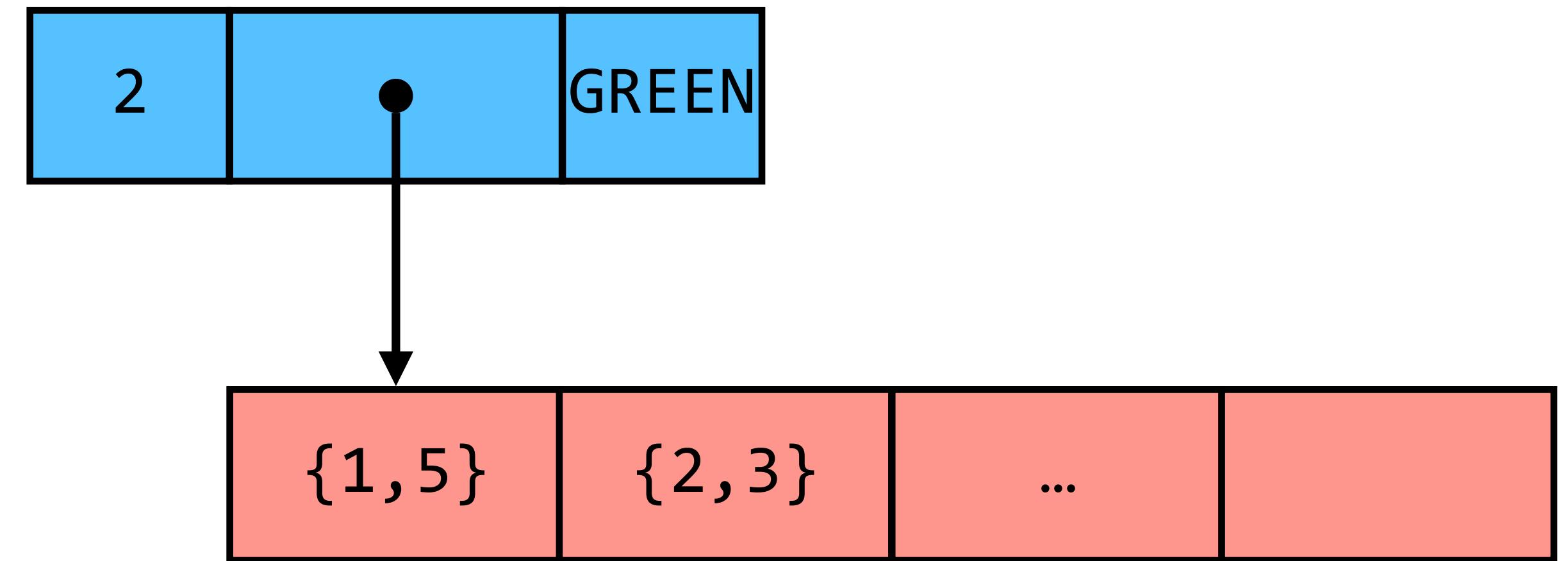


Poligonal com array na heap

```
#define MAX 100

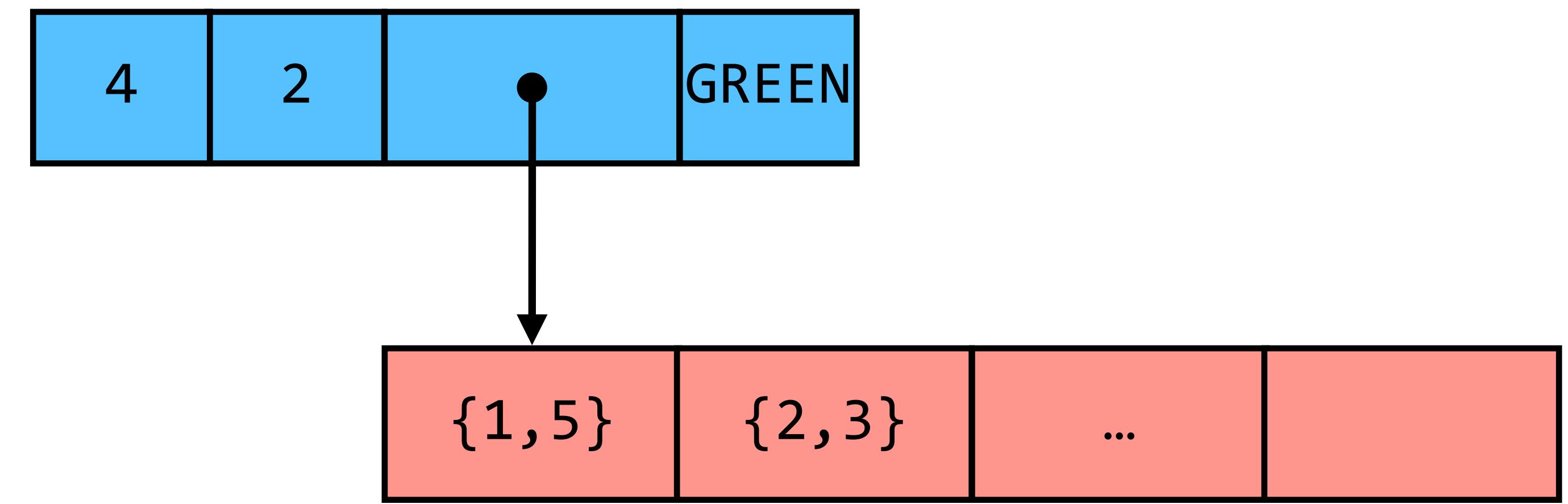
typedef struct {
    int tamanho;
    ponto *pontos;
    cor cor;
} poligonal;

int main() {
    ponto a = {1,5}, b = {2,3};
    poligonal l;
    l.tamanho = 0;
    l.cor = GREEN;
    l.pontos = malloc(sizeof(ponto) * MAX);
    estende(&l,a);
    estende(&l,b);
    free(l.pontos);
    return 0;
}
```



Poligonal com array dinâmico

```
typedef struct {  
    int capacidade;  
    int tamanho;  
    ponto *pontos;  
    cor cor;  
} poligonal;
```



aloca e liberta

```
#define MIN 1
```

```
int aloca(poligonal *l) {
    l->capacidade = MIN;
    l->tamanho = 0;
    l->pontos = malloc(l->capacidade * sizeof(ponto));
    if (l->pontos == NULL) return 1;
    else return 0;
}
```

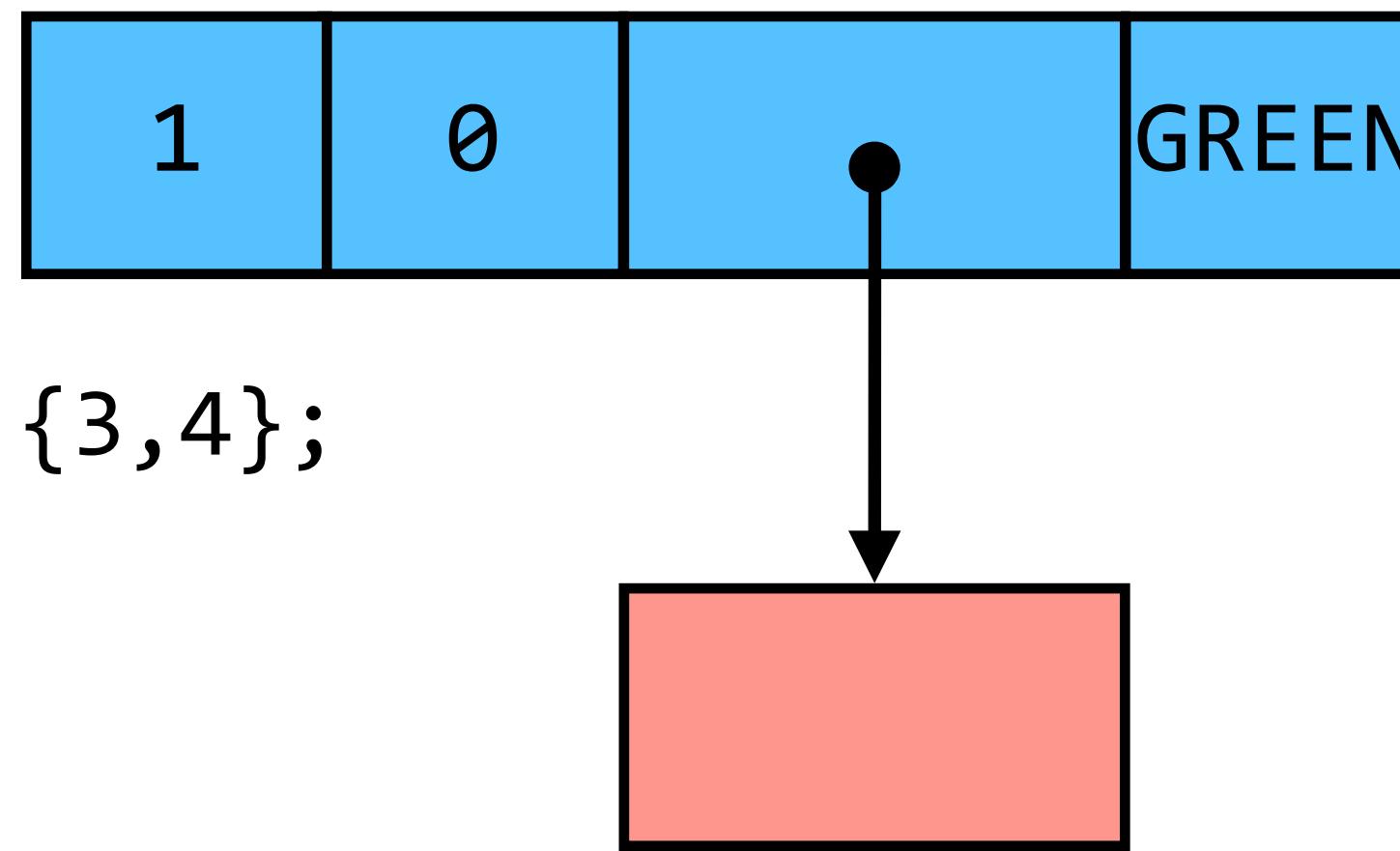
```
void liberta(poligonal *l) {
    free(l->pontos);
}
```

estende

```
int estende(poligonal *l, ponto p) {
    if (l->tamanho == l->capacidade) {
        ponto *new = malloc(2 * l->capacidade * sizeof(ponto));
        if (new == NULL) return 1;
        for (int i = 0; i < l->tamanho; i++) new[i] = l->pontos[i];
        free(l->pontos);
        l->pontos = new;
        l->capacidade *= 2;
    }
    l->pontos[l->tamanho] = p;
    l->tamanho++;
    return 0;
}
```

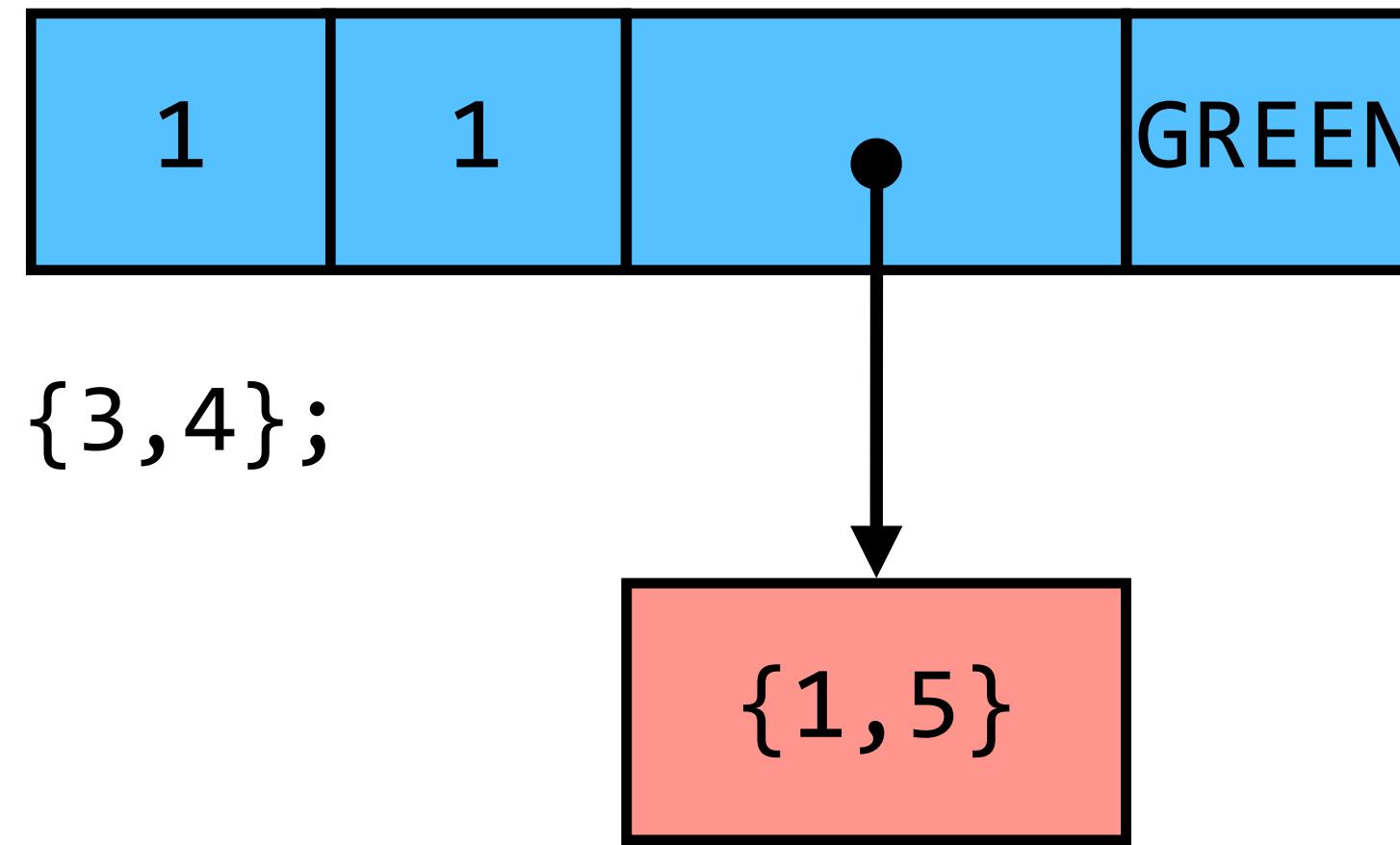
Poligonal com array dinâmico

```
int main() {
    ponto a = {1,5}, b = {2,3}, c = {3,4};
    poligonal l;
    aloca(&l);
    l.cor = GREEN;
    estende(&l,a);
    estende(&l,b);
    estende(&l,c);
    liberta(&l);
    return 0;
}
```



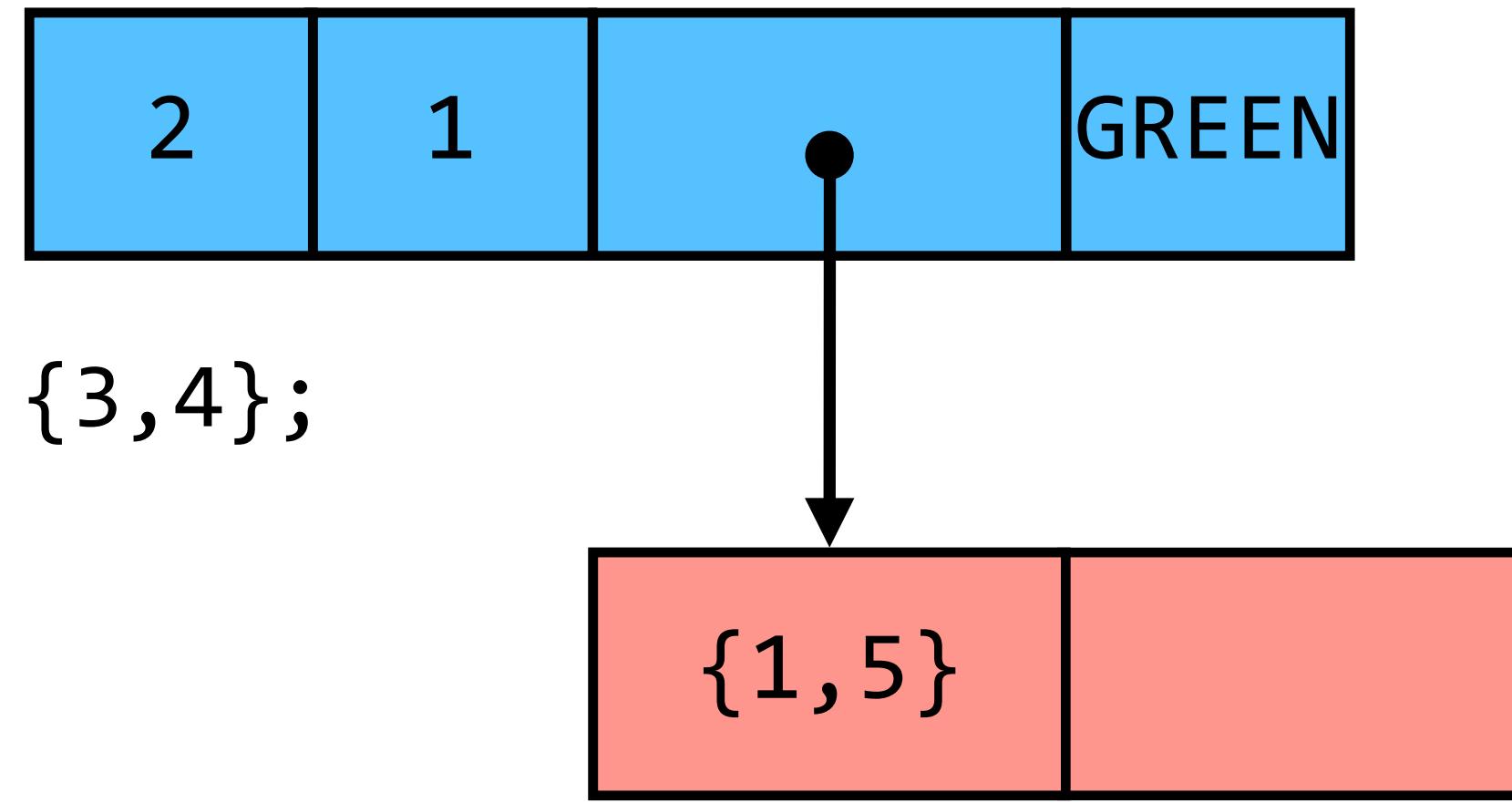
Poligonal com array dinâmico

```
int main() {
    ponto a = {1,5}, b = {2,3}, c = {3,4};
    poligonal l;
    aloca(&l);
    l.cor = GREEN;
    estende(&l,a);
    estende(&l,b);
    estende(&l,c);
    liberta(&l);
    return 0;
}
```



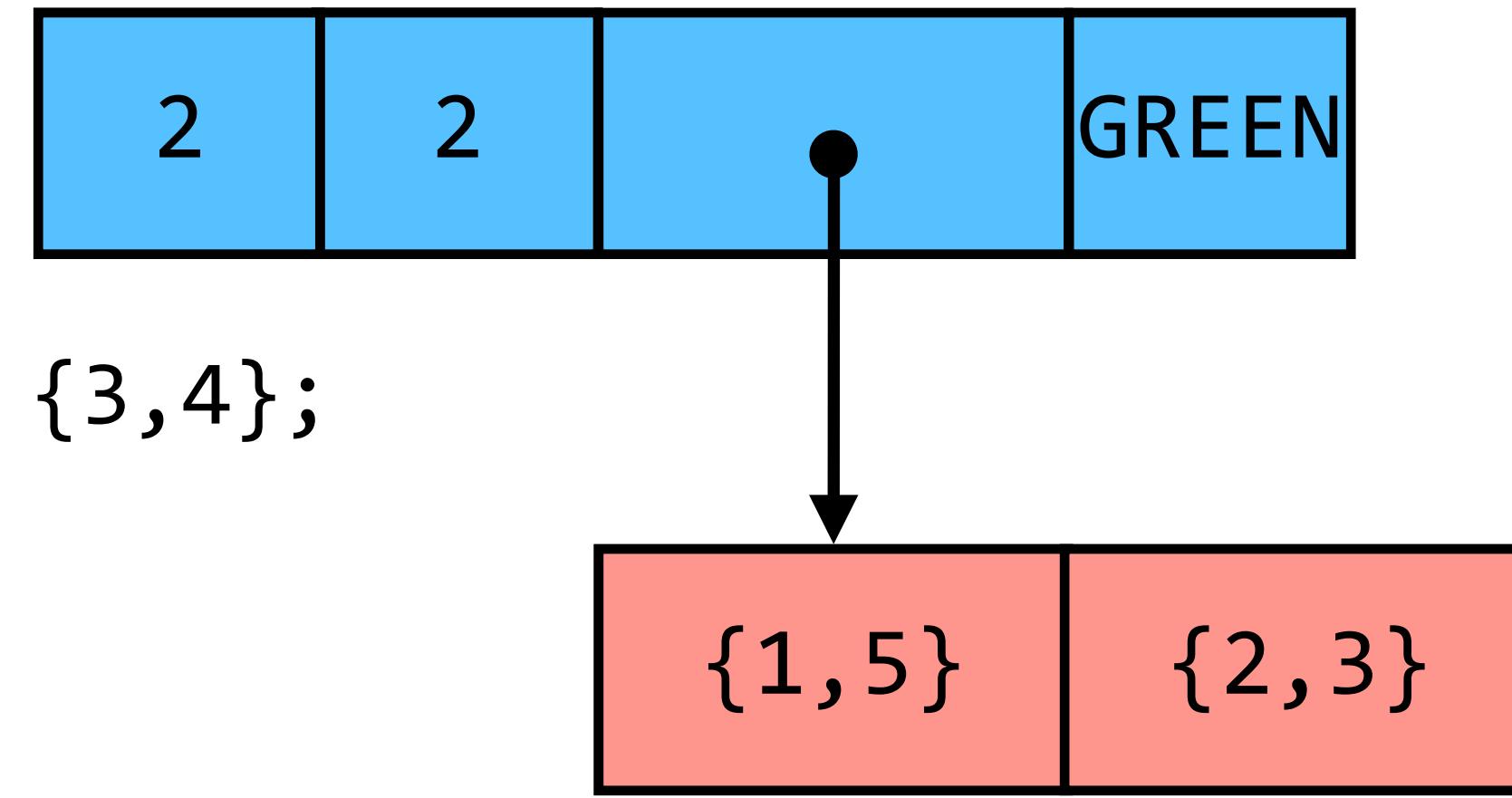
Poligonal com array dinâmico

```
int main() {
    ponto a = {1,5}, b = {2,3}, c = {3,4};
    poligonal l;
    aloca(&l);
    l.cor = GREEN;
    estende(&l,a);
    estende(&l,b);
    estende(&l,c);
    liberta(&l);
    return 0;
}
```



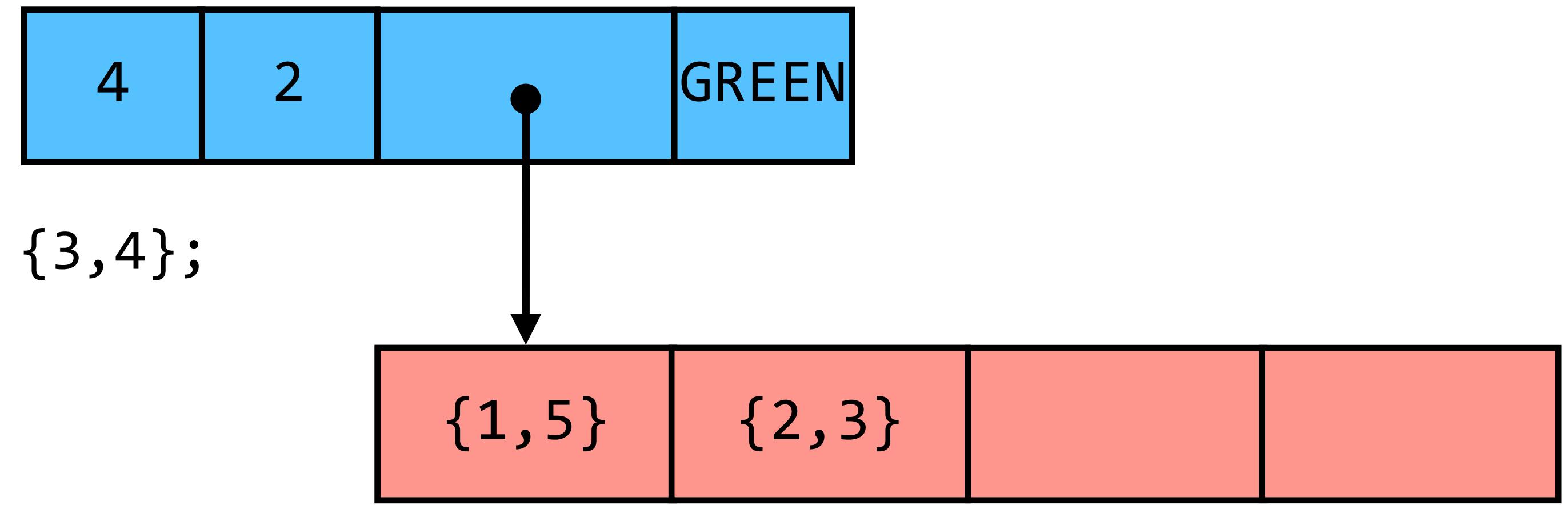
Poligonal com array dinâmico

```
int main() {
    ponto a = {1,5}, b = {2,3}, c = {3,4};
    poligonal l;
    aloca(&l);
    l.cor = GREEN;
    estende(&l,a);
    estende(&l,b);
    estende(&l,c);
    liberta(&l);
    return 0;
}
```



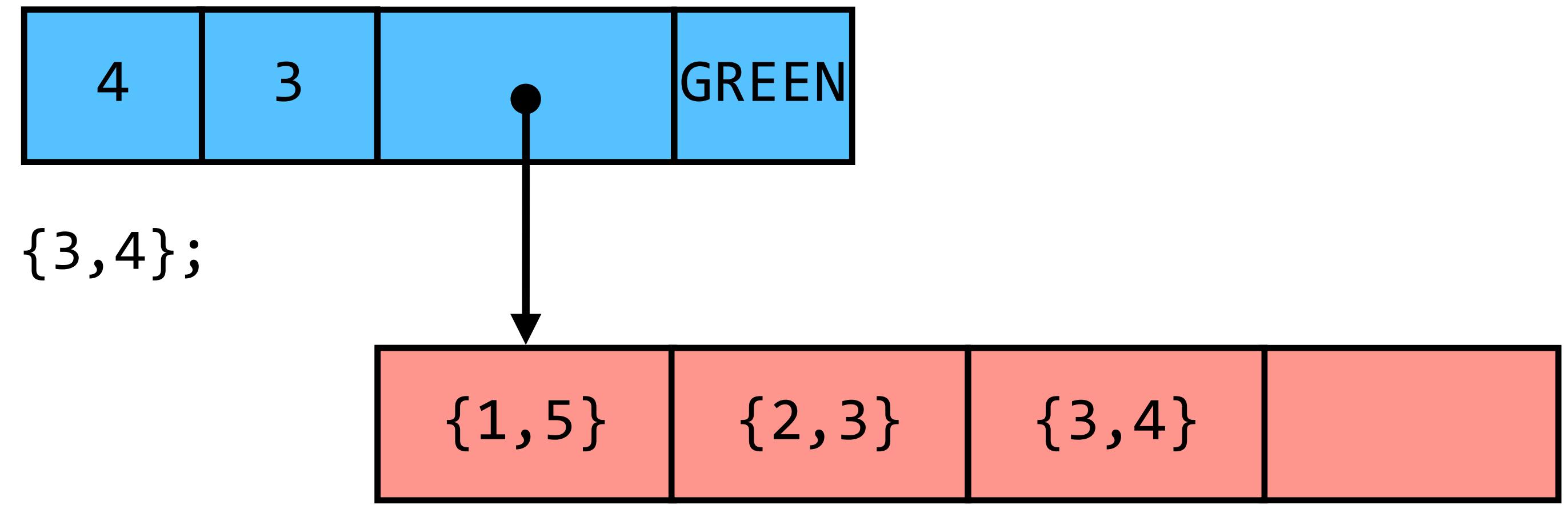
Poligonal com array dinâmico

```
int main() {
    ponto a = {1,5}, b = {2,3}, c = {3,4};
    poligonal l;
    aloca(&l);
    l.cor = GREEN;
    estende(&l,a);
    estende(&l,b);
    estende(&l,c);
    liberta(&l);
    return 0;
}
```



Poligonal com array dinâmico

```
int main() {
    ponto a = {1,5}, b = {2,3}, c = {3,4};
    poligonal l;
    aloca(&l);
    l.cor = GREEN;
    estende(&l,a);
    estende(&l,b);
    estende(&l,c);
    liberta(&l);
    return 0;
}
```



realloc

- A função realloc permite alterar o tamanho do espaço alocado
 - Recebe um apontador para o espaço actual e o novo tamanho desejado
 - Tenta primeiro alocar o espaço adicional depois do espaço actual
 - Se não for possível aloca numa nova zona da *heap*, transfere o conteúdo e liberta o espaço anterior
 - Se também não for possível devolve NULL
 - Neste caso não liberta o espaço anterior

estende

```
int estende(poligonal *l, ponto p) {
    if (l->tamanho == l->capacidade) {
        ponto *new = realloc(l->pontos, 2 * l->capacidade * sizeof(ponto));
        if (new == NULL) return 1;
        l->pontos = new;
        l->capacidade *= 2;
    }
    l->pontos[l->tamanho] = p;
    l->tamanho++;
    return 0;
}
```

#19 Aproximadamente, quantos reallocs serão feitos?

```
#define N ...  
  
int main() {  
    ponto p;  
    poligonal l;  
    aloca(&l);  
    for (int i = 0; i < N; i++) {  
        p.x = i; p.y = i;  
        estende(&l, p);  
    }  
    liberta(&l);  
    return 0;  
}
```

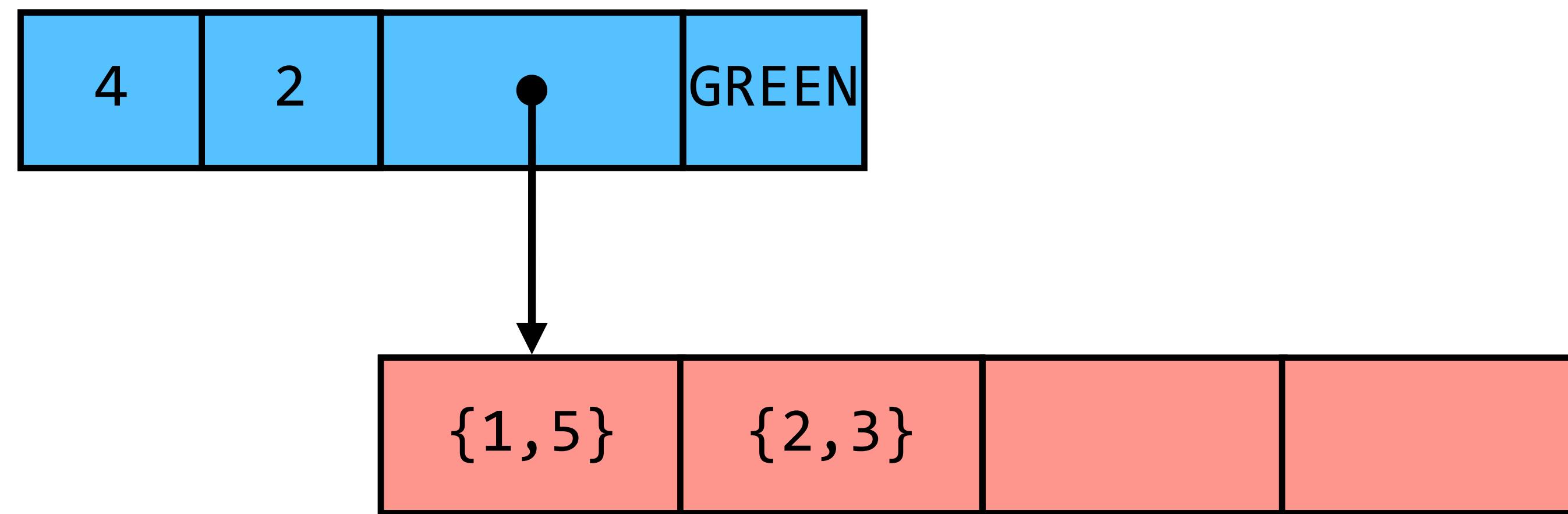


Arrays dinâmicos

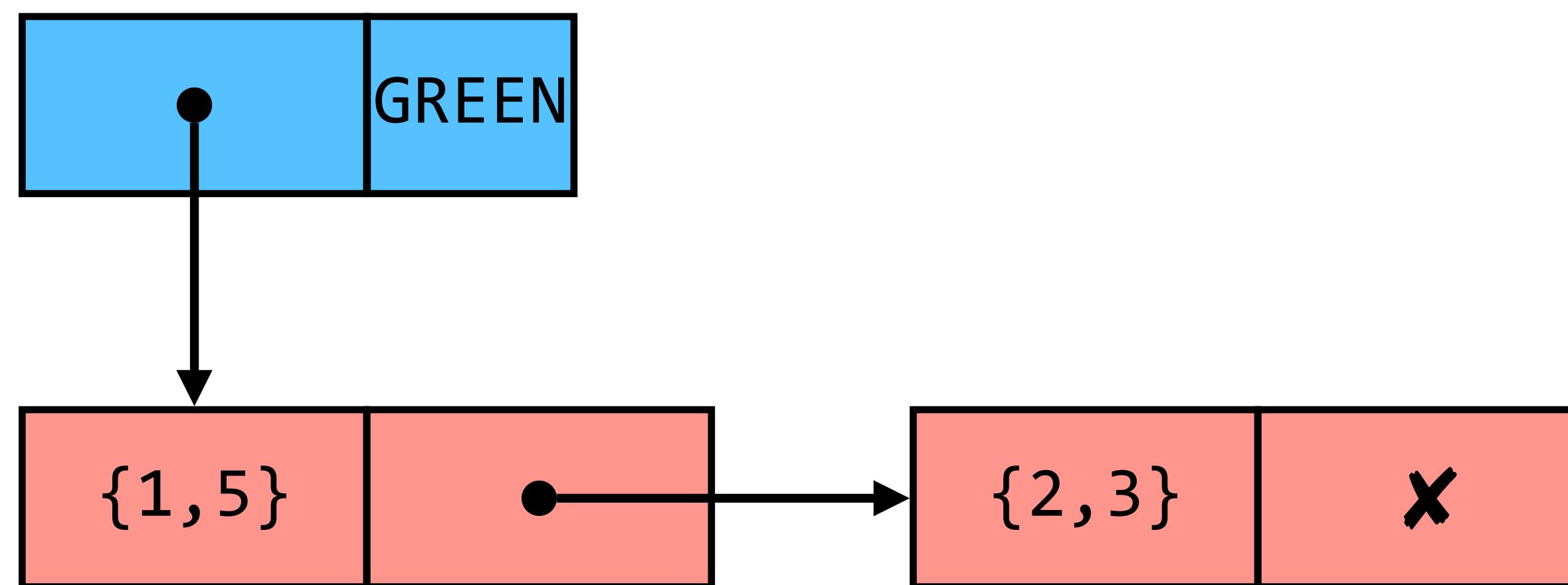
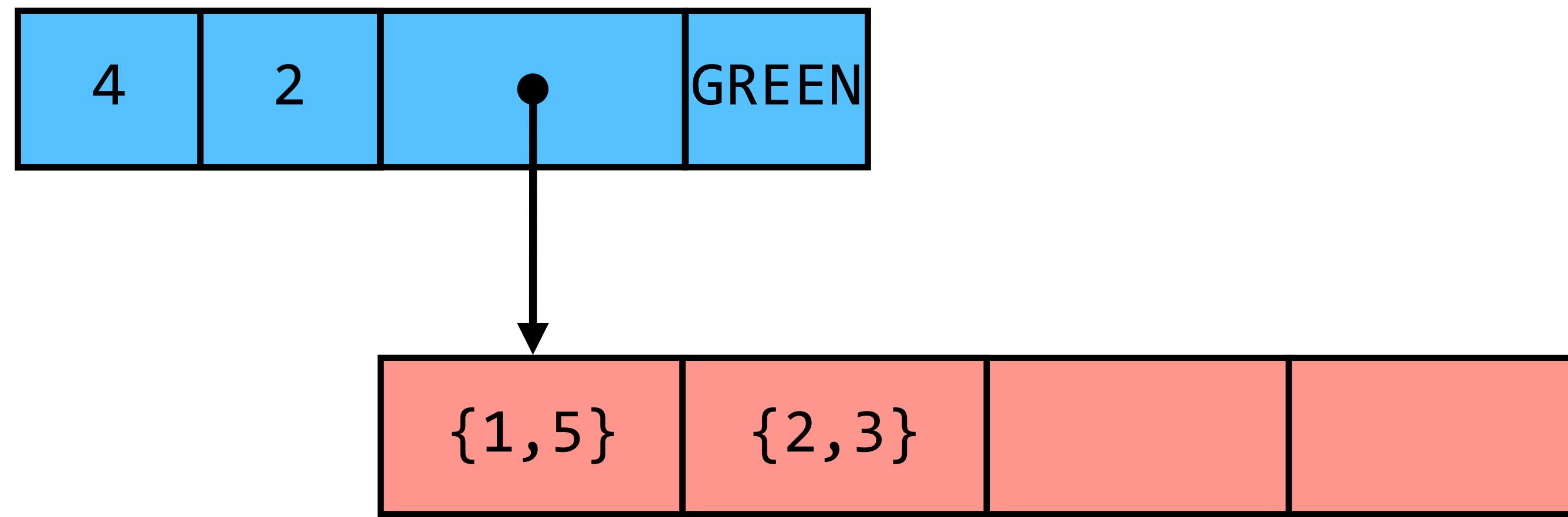
- Permitem implementar arrays de tamanho variável
- Com operações de leitura e escrita muito eficientes, tal como nos arrays normais
- Mas com uma operação ocasional de redimensionamento pouco eficiente
- E também podem ter muito espaço desperdiçado
- Existem outras estruturas de dados dinâmicas com operações cuja eficiência pode ser mais previsível
- E com espaço ocupado proporcional ao tamanho do conteúdo

Arrays dinâmicos vs Listas ligadas

Arrays dinâmicos vs Listas ligadas



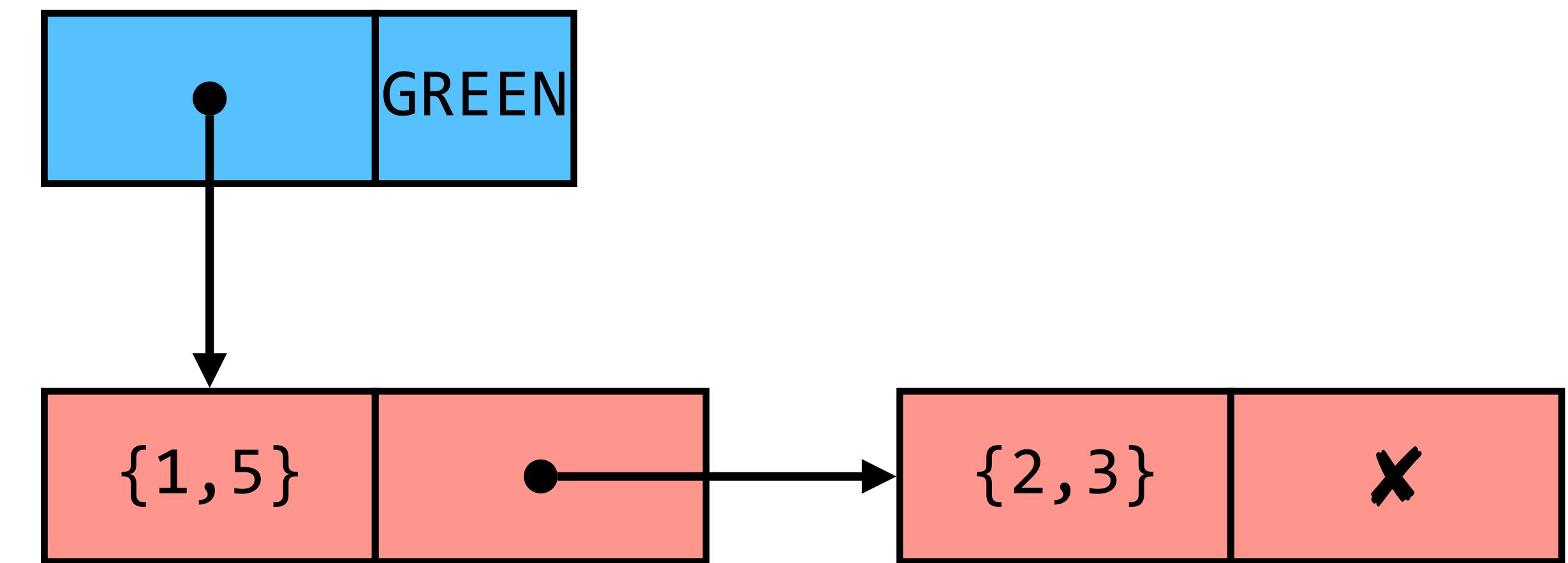
Arrays dinâmicos vs Listas ligadas



Poligonal com lista ligada

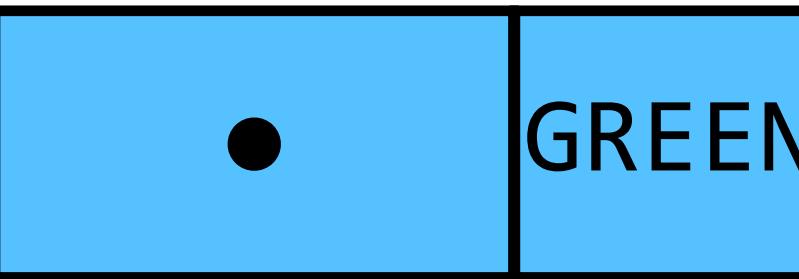
```
typedef struct lponto_no {  
    ponto ponto;  
    struct lponto_no *prox;  
} *lponto;
```

```
typedef struct {  
    lponto pontos;  
    cor cor;  
} poligonal;
```



Aula 11

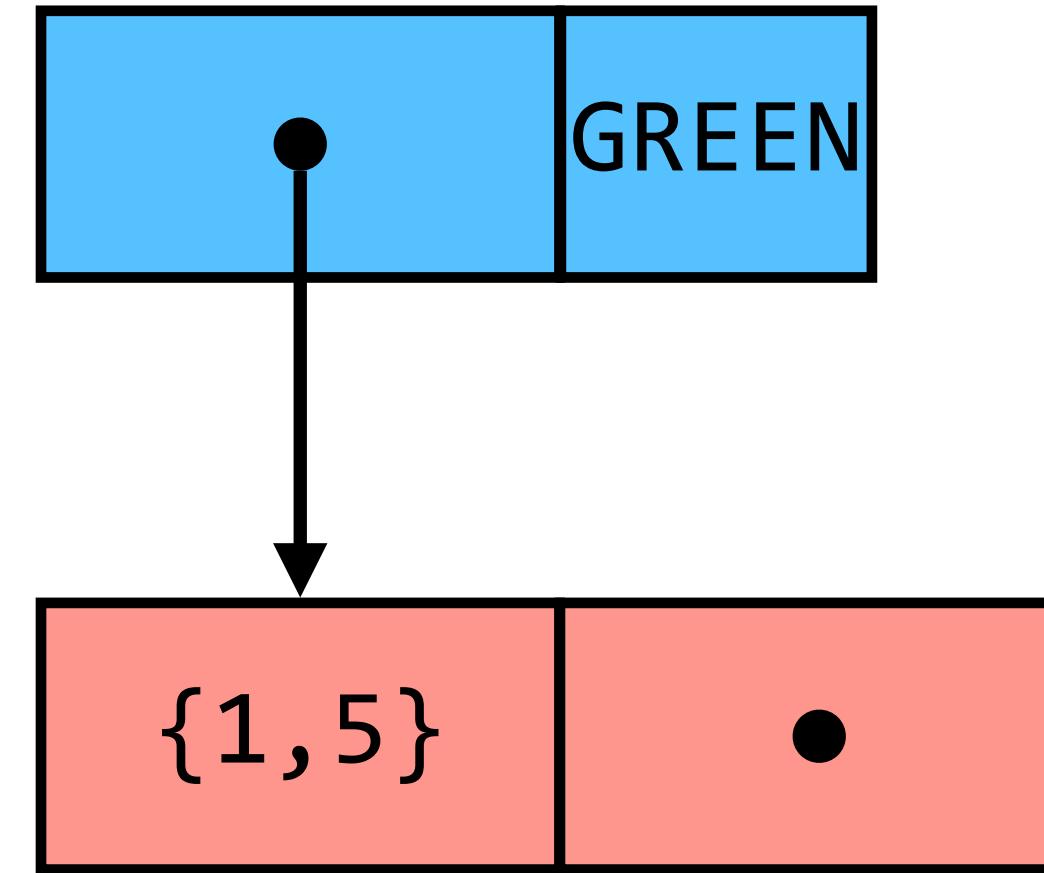
Poligonal com lista ligada



```
int main() {
    ponto a = {1,5}, b = {2,3};
    poligonal l;
    l.cor = GREEN;
    l.pontos = malloc(sizeof(struct lponto_no));
    l.pontos->ponto = a;
    l.pontos->prox = malloc(sizeof(struct lponto_no));
    l.pontos->prox->ponto = b;
    l.pontos->prox->prox = NULL;
    free(l.pontos->prox);
    free(l.pontos);
    return 0;
}
```

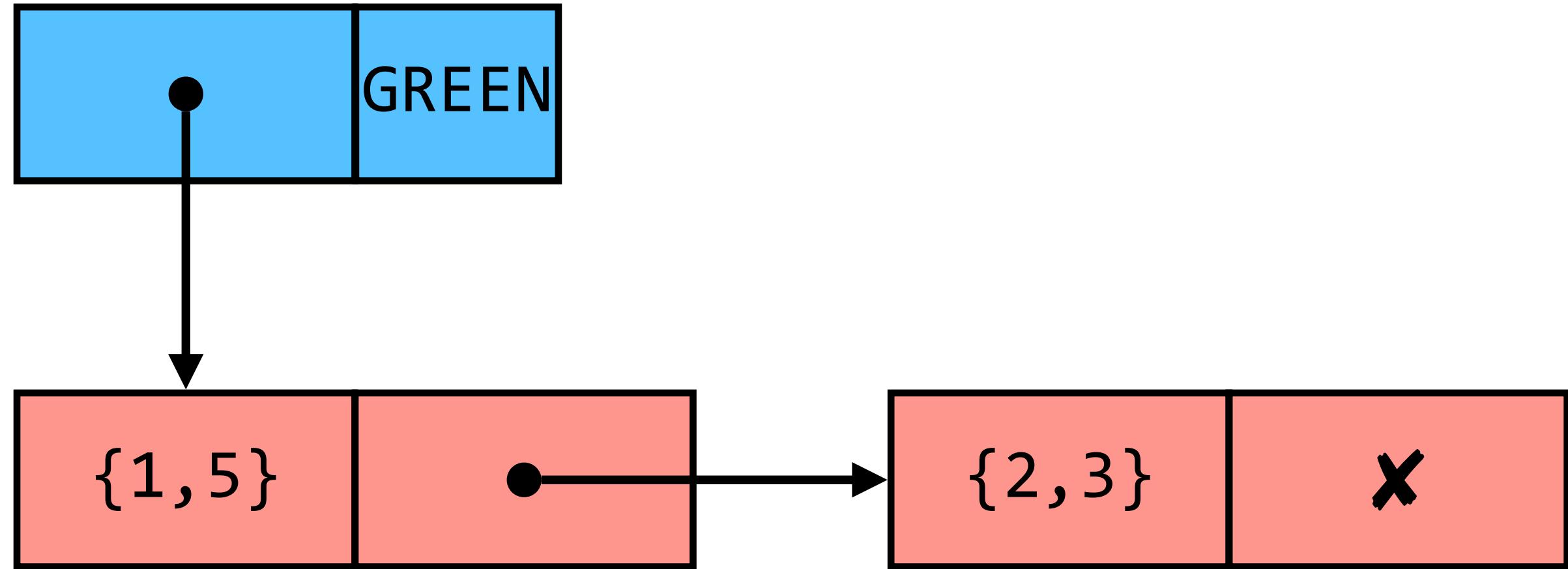
Poligonal com lista ligada

```
int main() {
    ponto a = {1,5}, b = {2,3};
    poligonal l;
    l.cor = GREEN;
    l.pontos = malloc(sizeof(struct lponto_no));
    l.pontos->ponto = a;
    l.pontos->prox = malloc(sizeof(struct lponto_no));
    l.pontos->prox->ponto = b;
    l.pontos->prox->prox = NULL;
    free(l.pontos->prox);
    free(l.pontos);
    return 0;
}
```



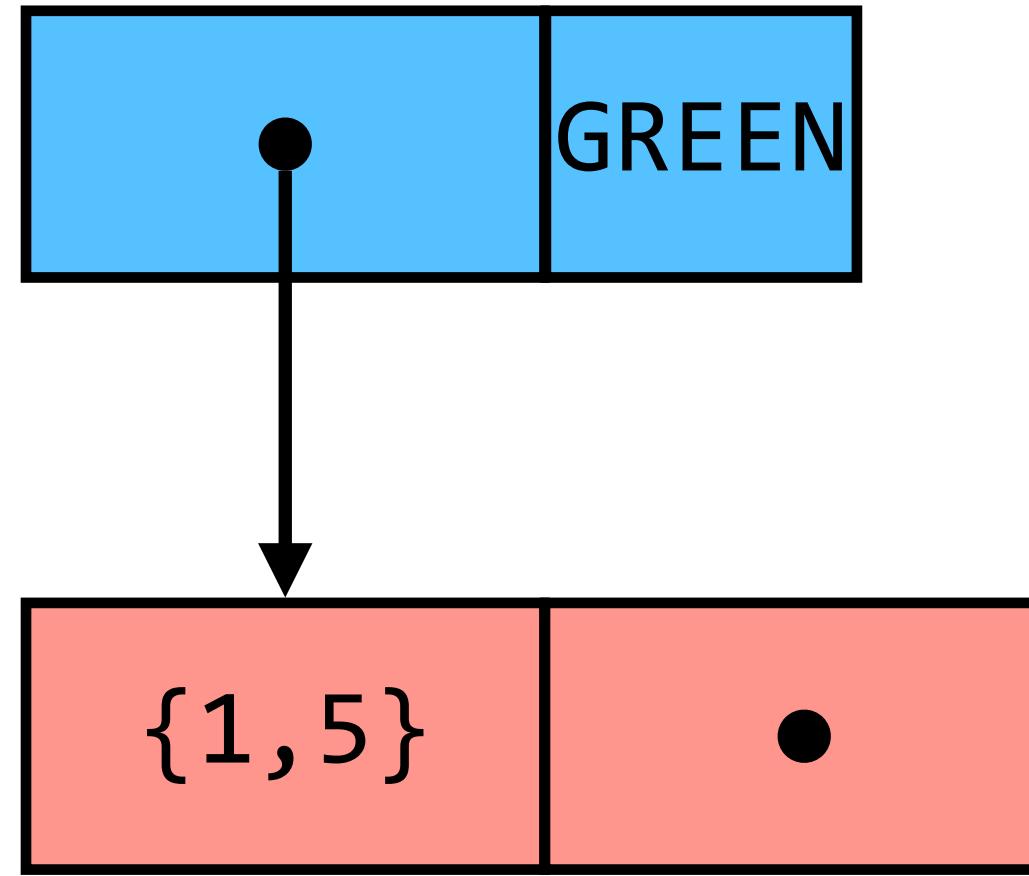
Poligonal com lista ligada

```
int main() {
    ponto a = {1,5}, b = {2,3};
    poligonal l;
    l.cor = GREEN;
    l.pontos = malloc(sizeof(struct lponto_no));
    l.pontos->ponto = a;
    l.pontos->prox = malloc(sizeof(struct lponto_no));
    l.pontos->prox->ponto = b;
    l.pontos->prox->prox = NULL;
    free(l.pontos->prox);
    free(l.pontos);
    return 0;
}
```

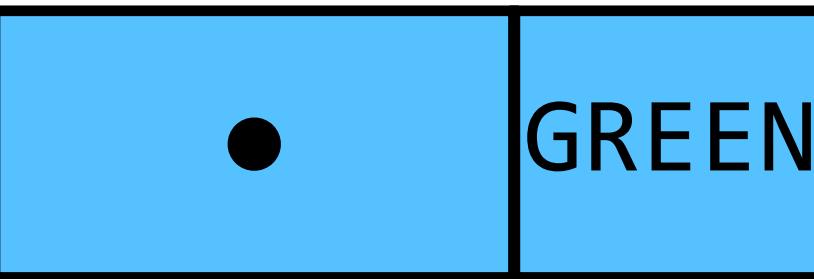


Poligonal com lista ligada

```
int main() {
    ponto a = {1,5}, b = {2,3};
    poligonal l;
    l.cor = GREEN;
    l.pontos = malloc(sizeof(struct lponto_no));
    l.pontos->ponto = a;
    l.pontos->prox = malloc(sizeof(struct lponto_no));
    l.pontos->prox->ponto = b;
    l.pontos->prox->prox = NULL;
    free(l.pontos->prox);
    free(l.pontos);
    return 0;
}
```

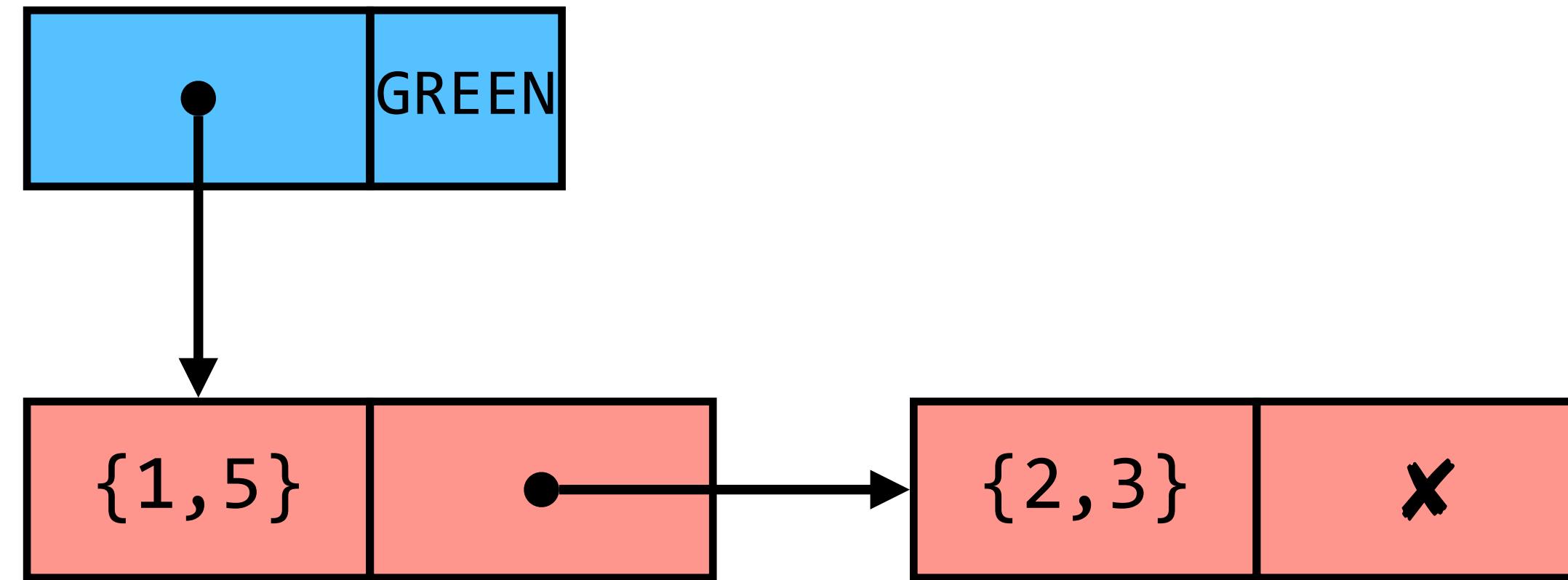


Poligonal com lista ligada



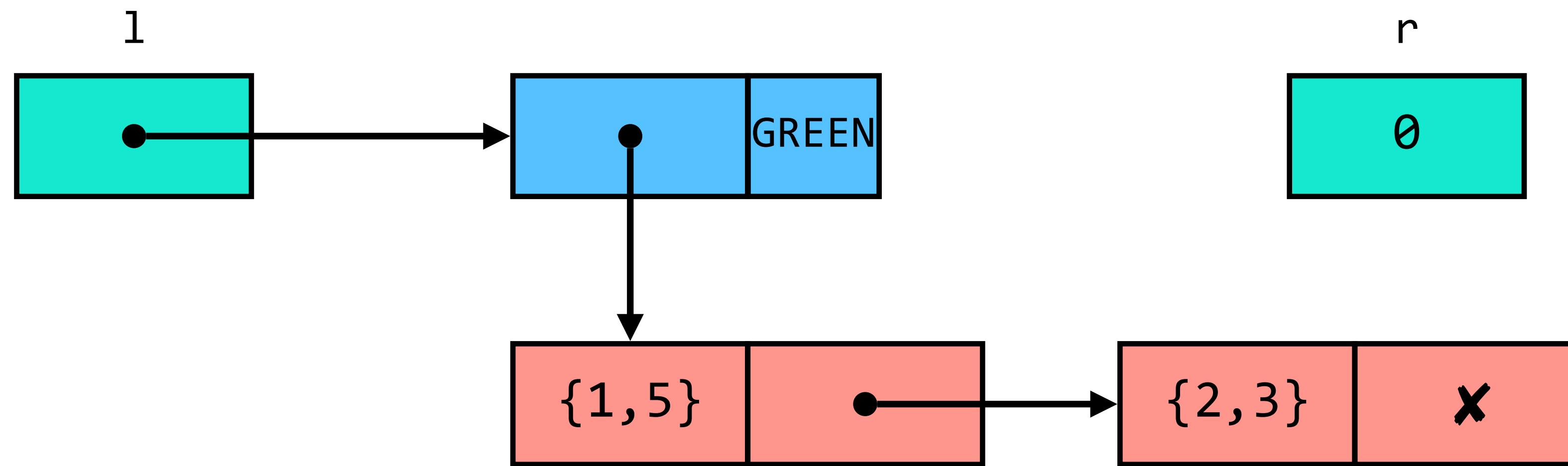
```
int main() {
    ponto a = {1,5}, b = {2,3};
    poligonal l;
    l.cor = GREEN;
    l.pontos = malloc(sizeof(struct lponto_no));
    l.pontos->ponto = a;
    l.pontos->prox = malloc(sizeof(struct lponto_no));
    l.pontos->prox->ponto = b;
    l.pontos->prox->prox = NULL;
    free(l.pontos->prox);
    free(l.pontos);
    return 0;
}
```

tamanho



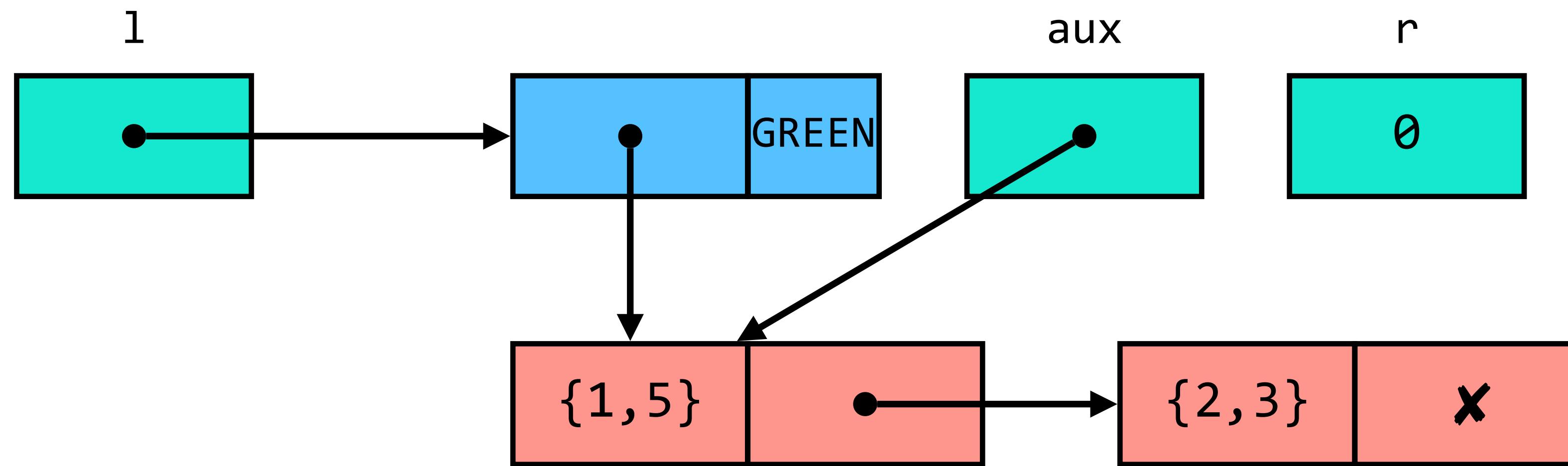
```
int tamanho(poligonal *l) {  
    int r = 0;  
    for (lponto aux = l->pontos; aux != NULL; aux = aux->prox) r++;  
    return r;  
}
```

tamanho



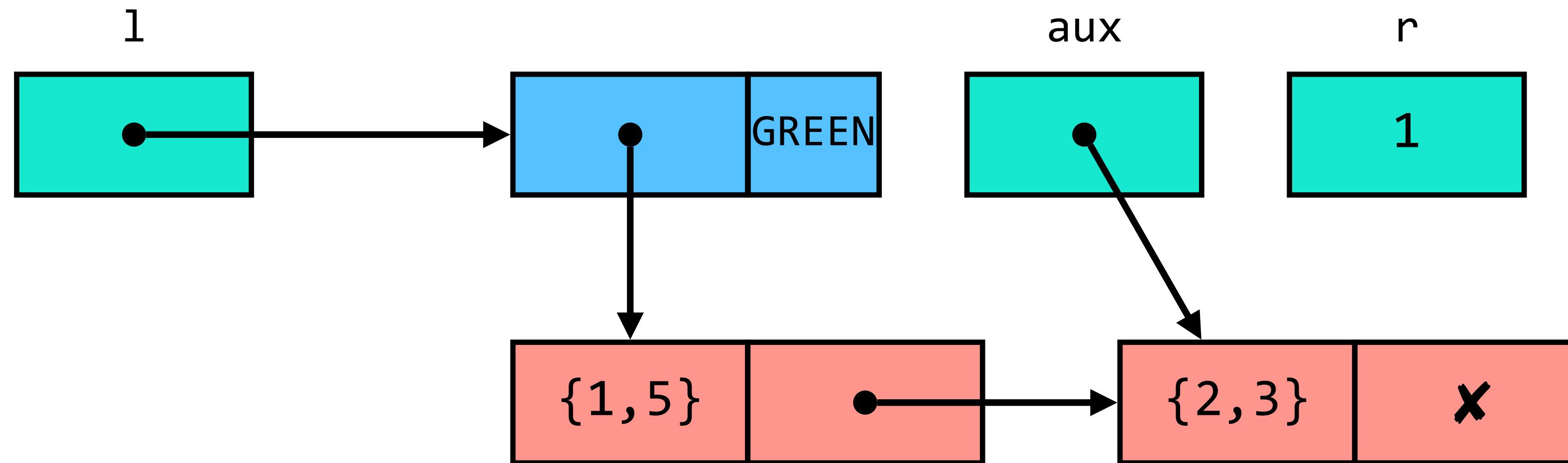
```
int tamanho(poligonal *l) {  
    int r = 0;  
    for (lponto aux = l->pontos; aux != NULL; aux = aux->prox) r++;  
    return r;  
}
```

tamanho



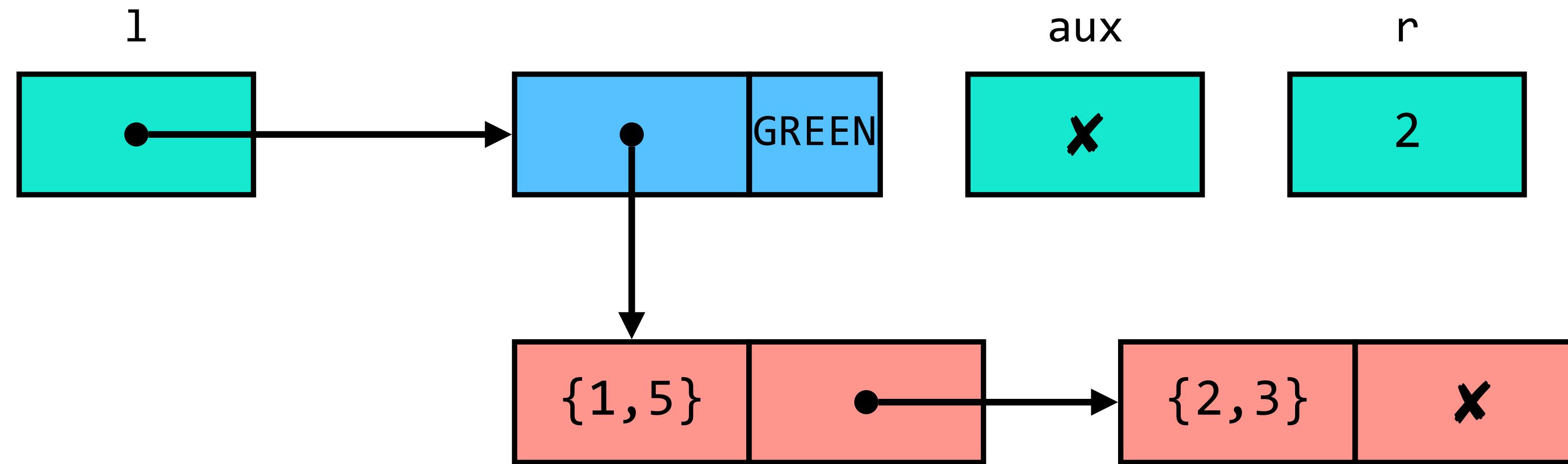
```
int tamanho(poligonal *l) {  
    int r = 0;  
    for (lponto aux = l->pontos; aux != NULL; aux = aux->prox) r++;  
    return r;  
}
```

tamanho



```
int tamanho(poligonal *l) {  
    int r = 0;  
    for (lponto aux = l->pontos; aux != NULL; aux = aux->prox) r++;  
    return r;  
}
```

tamanho



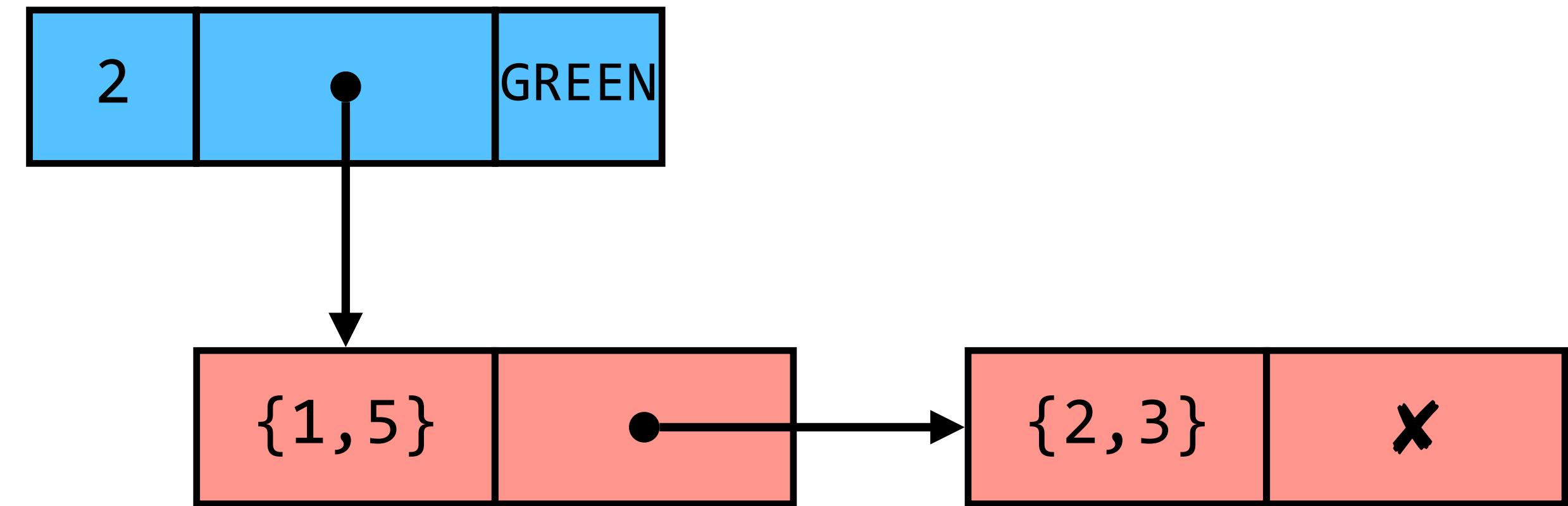
```
int tamanho(poligonal *l) {  
    int r = 0;  
    for (lponto aux = l->pontos; aux != NULL; aux = aux->prox) r++;  
    return r;  
}
```



Poligonal com lista ligada

```
typedef struct lponto_no {  
    ponto ponto;  
    struct lponto_no *prox;  
} *lponto;
```

```
typedef struct {  
    int tamanho;  
    lponto pontos;  
    cor cor;  
} poligonal;
```



tamanho

```
int tamanho(polygonal *l) {  
    return l->tamanho;  
}
```

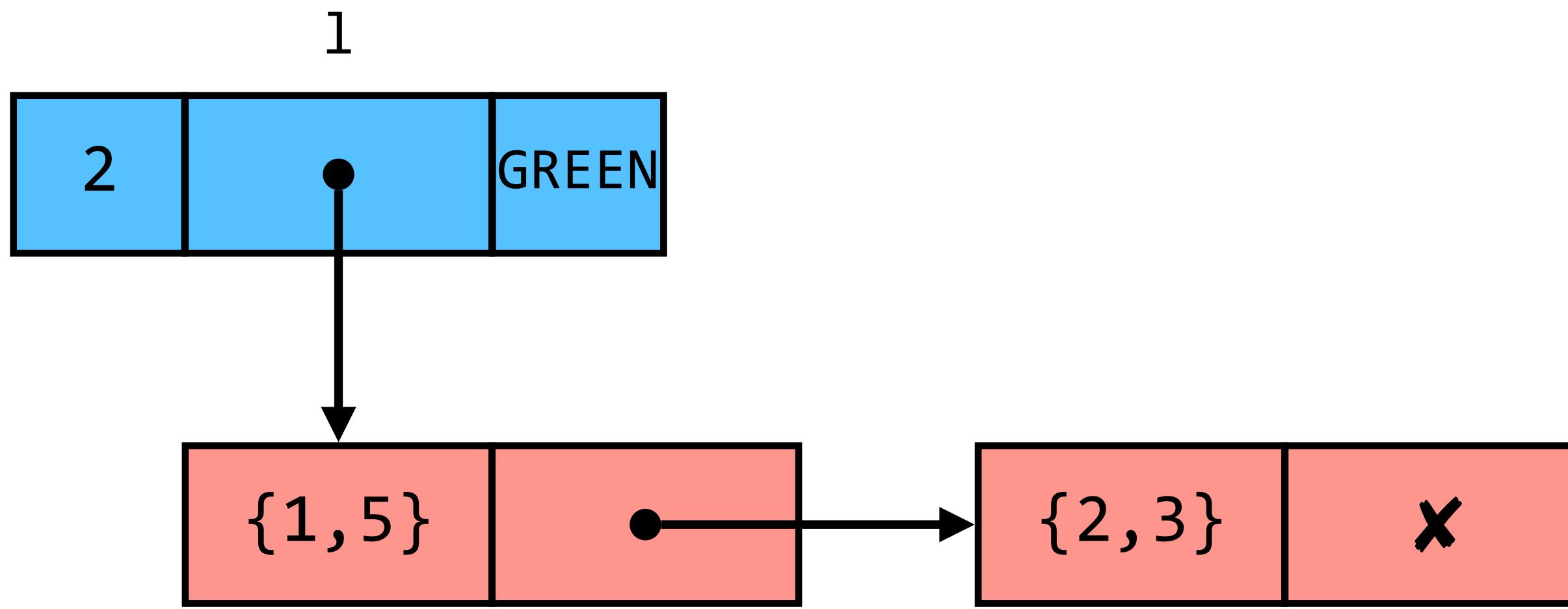
comprimento

```
float comprimento(poligonal *l) {
    float r = 0;
    if (l->pontos == NULL) return r;
    for (lponto aux = l->pontos; aux->prox != NULL; aux = aux->prox)
        r += dist(aux->ponto, aux->prox->ponto);
    return r;
}
```

estende

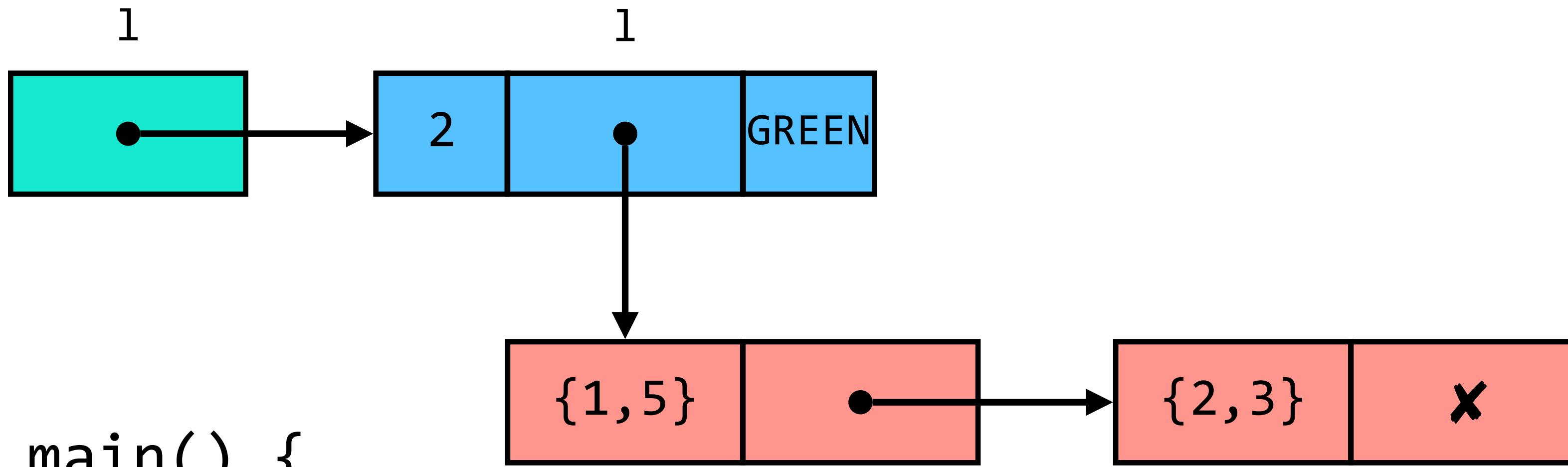
```
int estende(poligonal *l, ponto p) {  
    ...  
}
```

estende



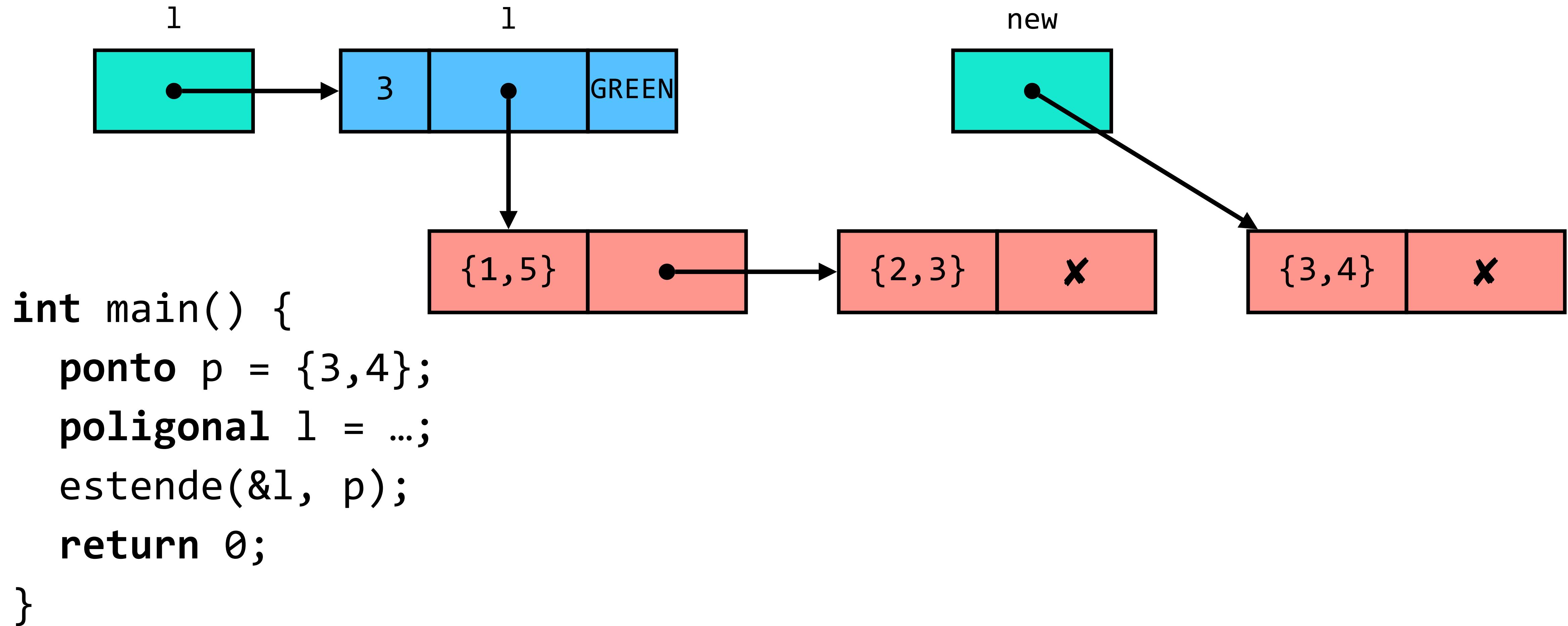
```
int main() {
    ponto p = {3,4};
    poligonal l = ...;
    estende(&l, p);
    return 0;
}
```

estende

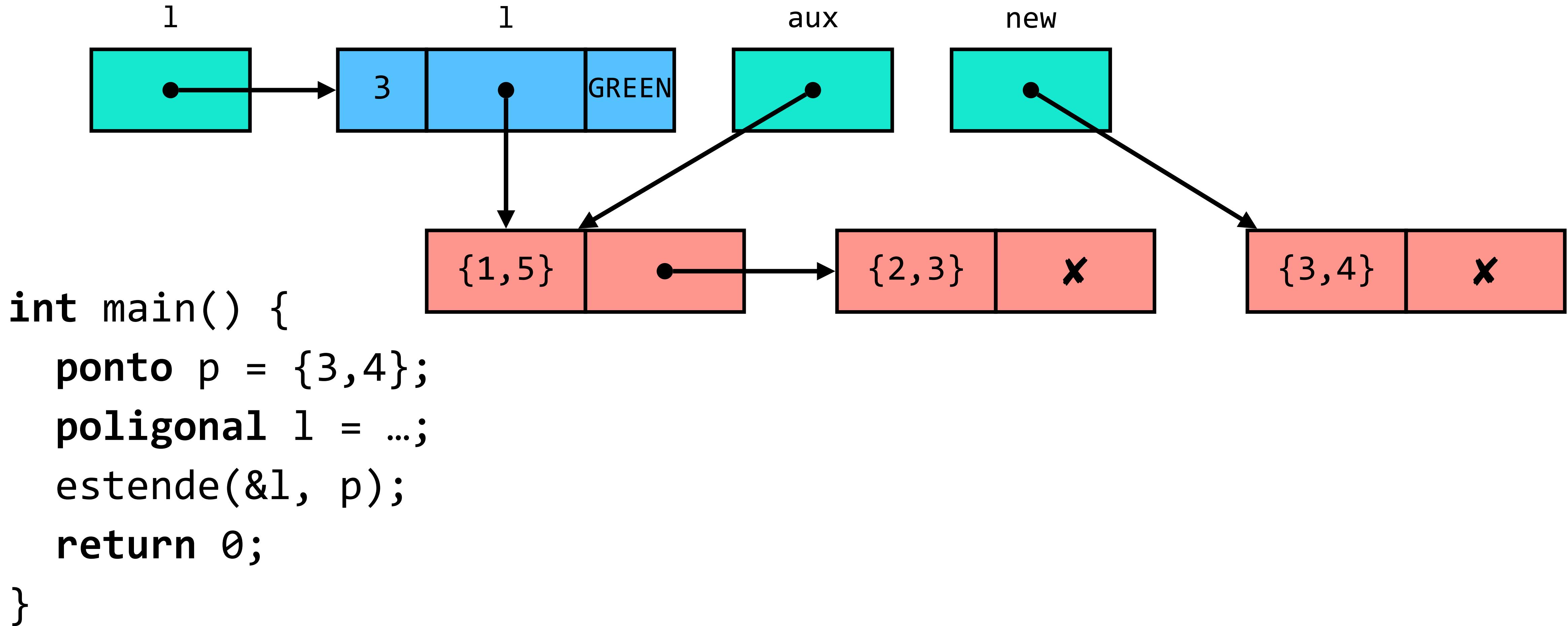


```
int main() {
    ponto p = {3,4};
    poligonal l = ...;
    estende(&l, p);
    return 0;
}
```

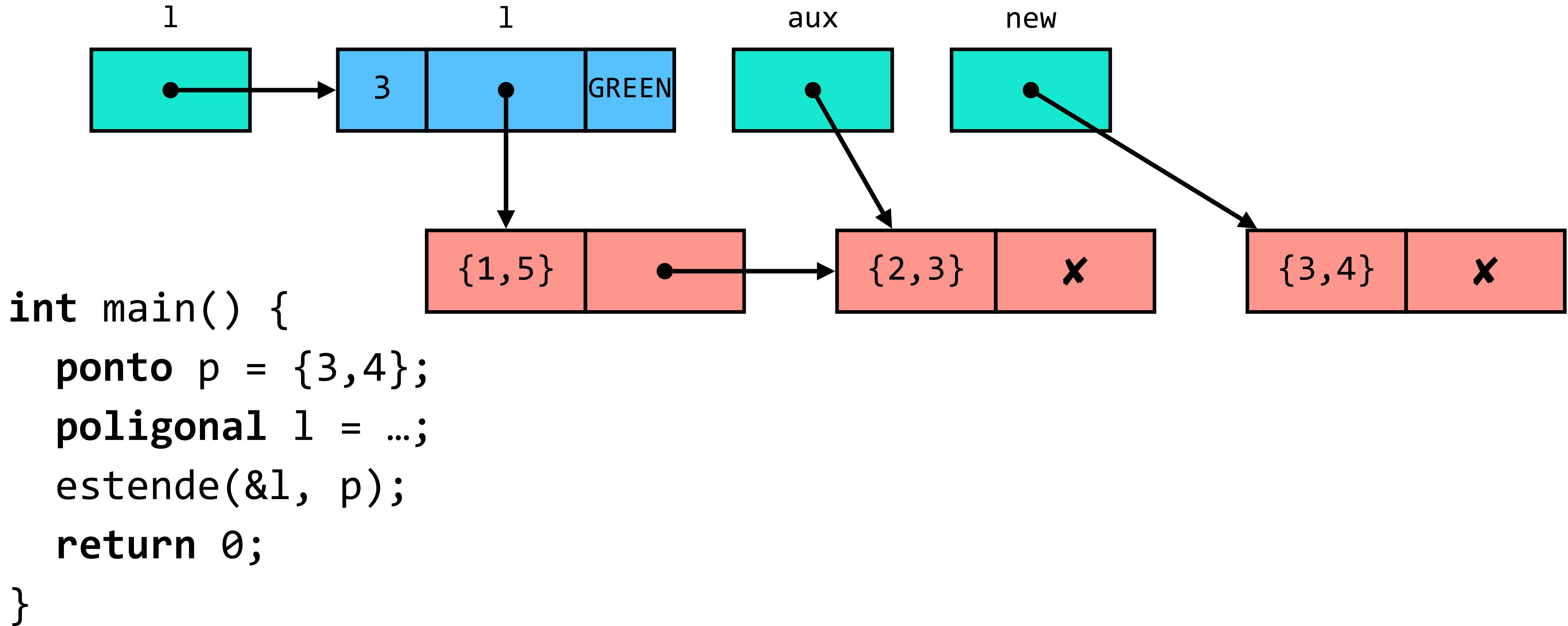
estende



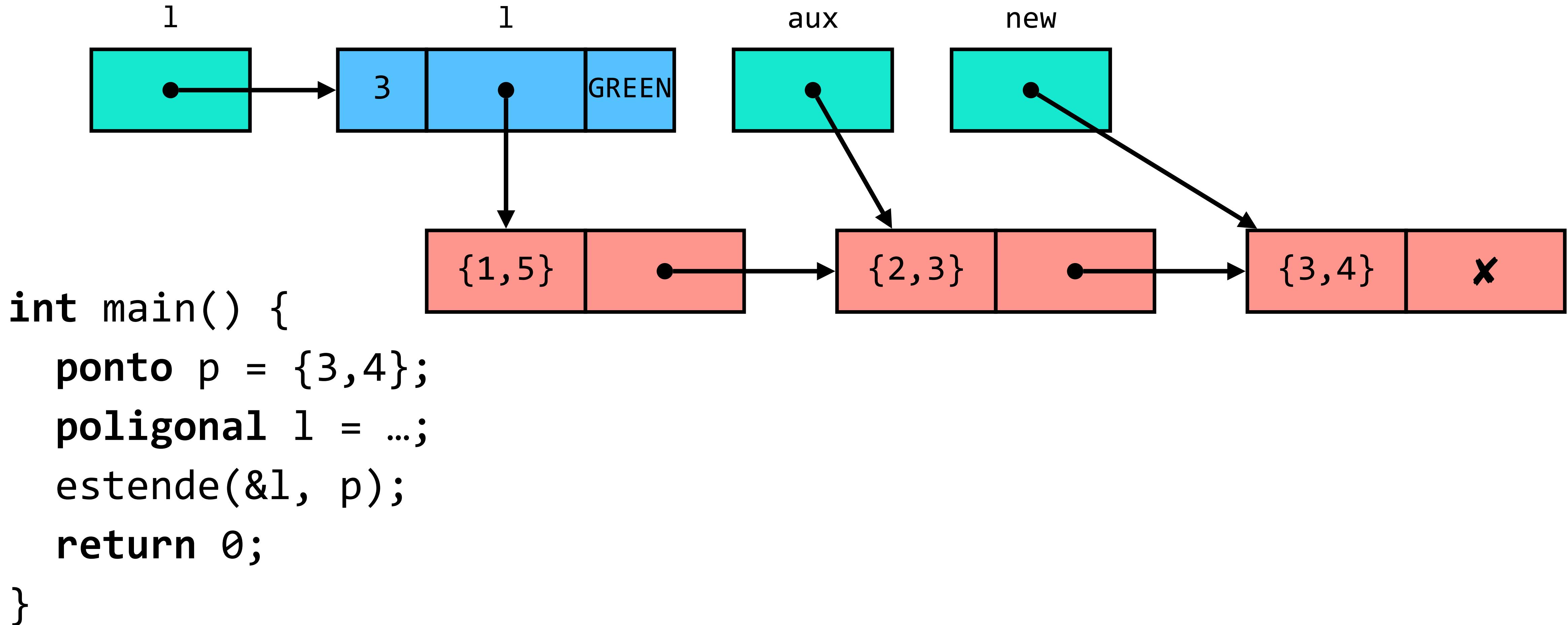
estende



estende

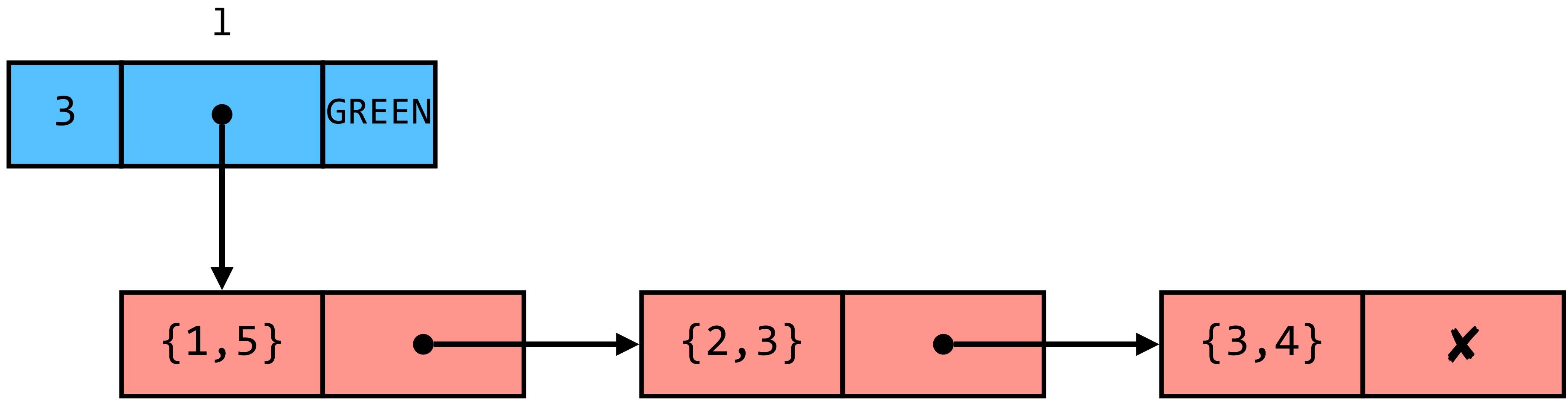


estende

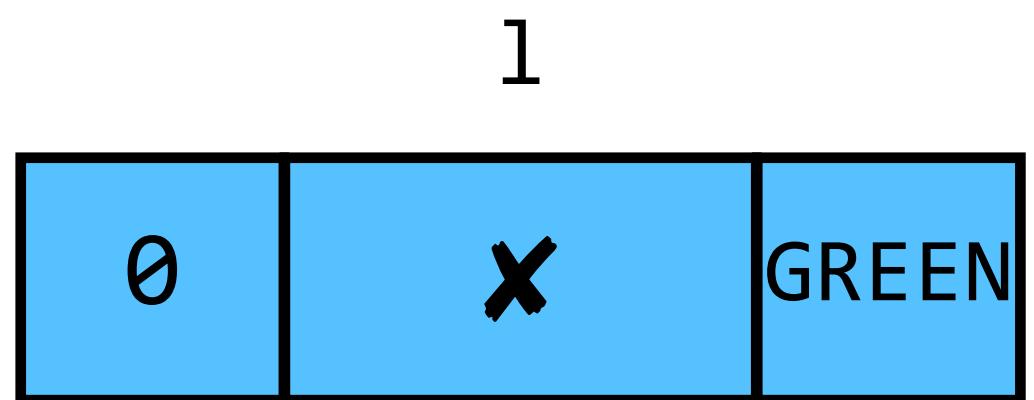


estende

```
int main() {  
    ponto p = {3,4};  
    poligonal l = ...;  
    estende(&l, p);  
    return 0;  
}
```

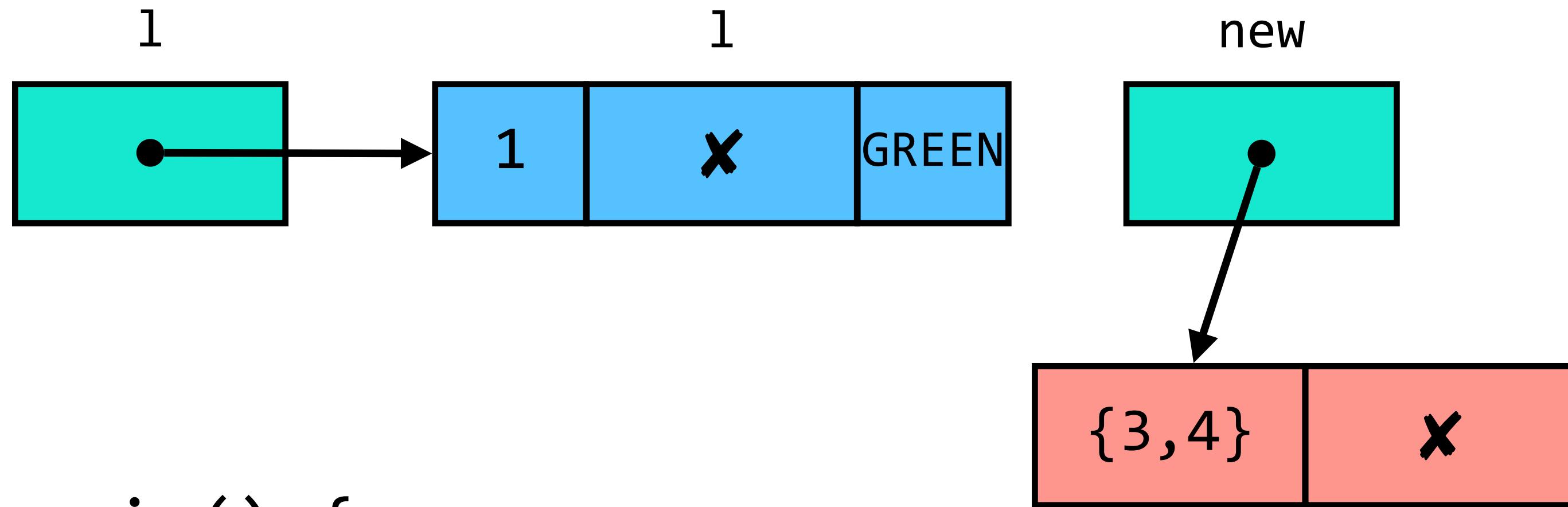


estende



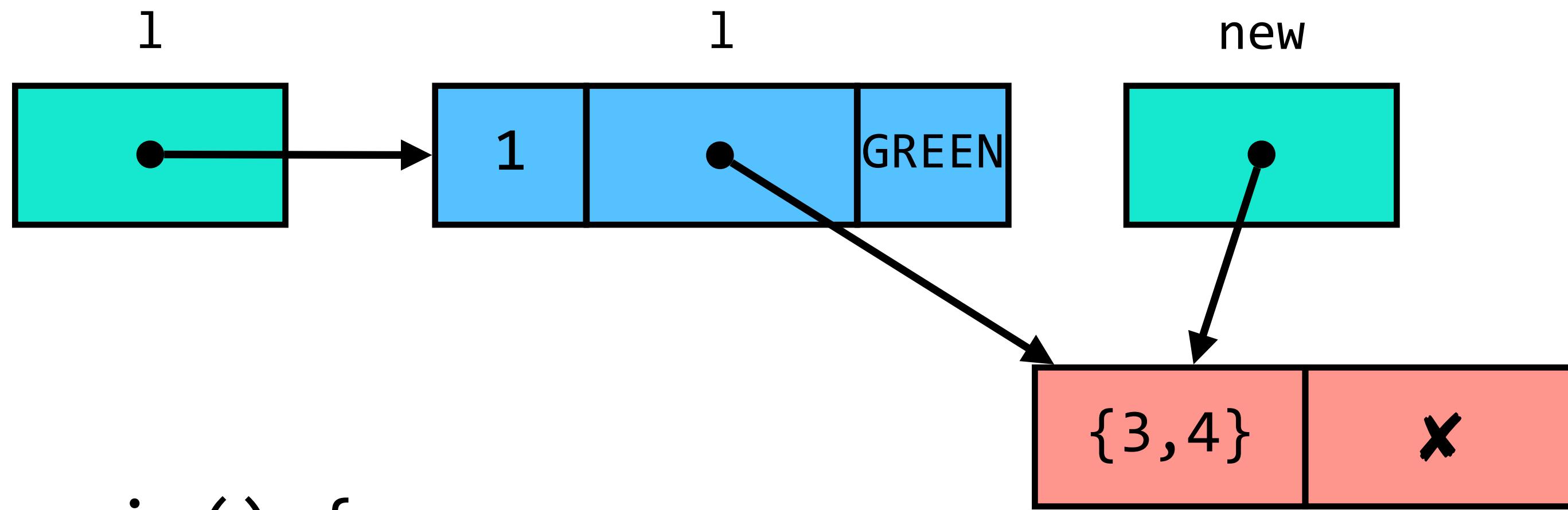
```
int main() {
    ponto p = {3,4};
    poligonal l = {0, NULL};
    estende(&l, p);
    return 0;
}
```

estende



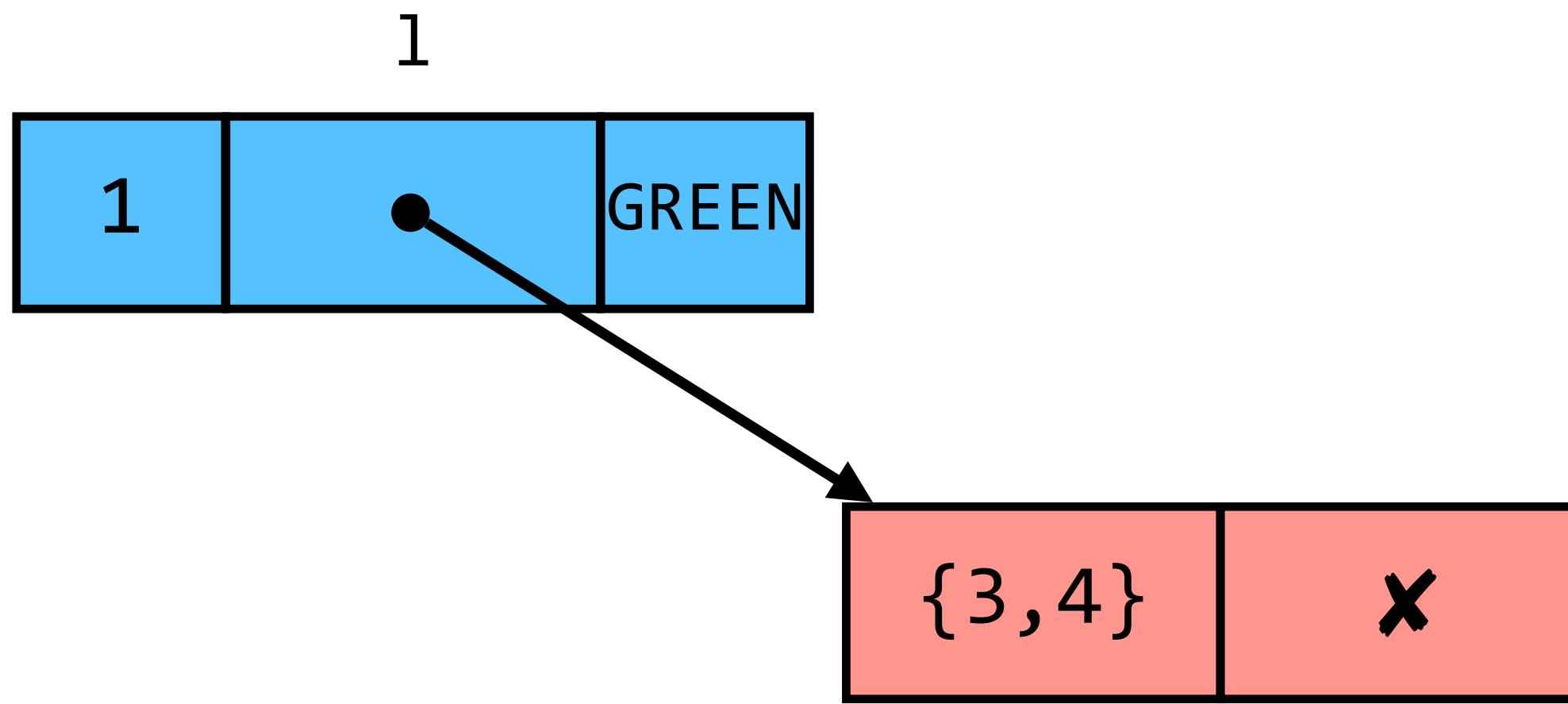
```
int main() {
    ponto p = {3,4};
    poligonal l = {0, NULL};
    estende(&l, p);
    return 0;
}
```

estende



```
int main() {
    ponto p = {3,4};
    poligonal l = {0, NULL};
    estende(&l, p);
    return 0;
}
```

estende



```
int main() {
    ponto p = {3,4};
    poligonal l = {0, NULL};
    estende(&l, p);
    return 0;
}
```

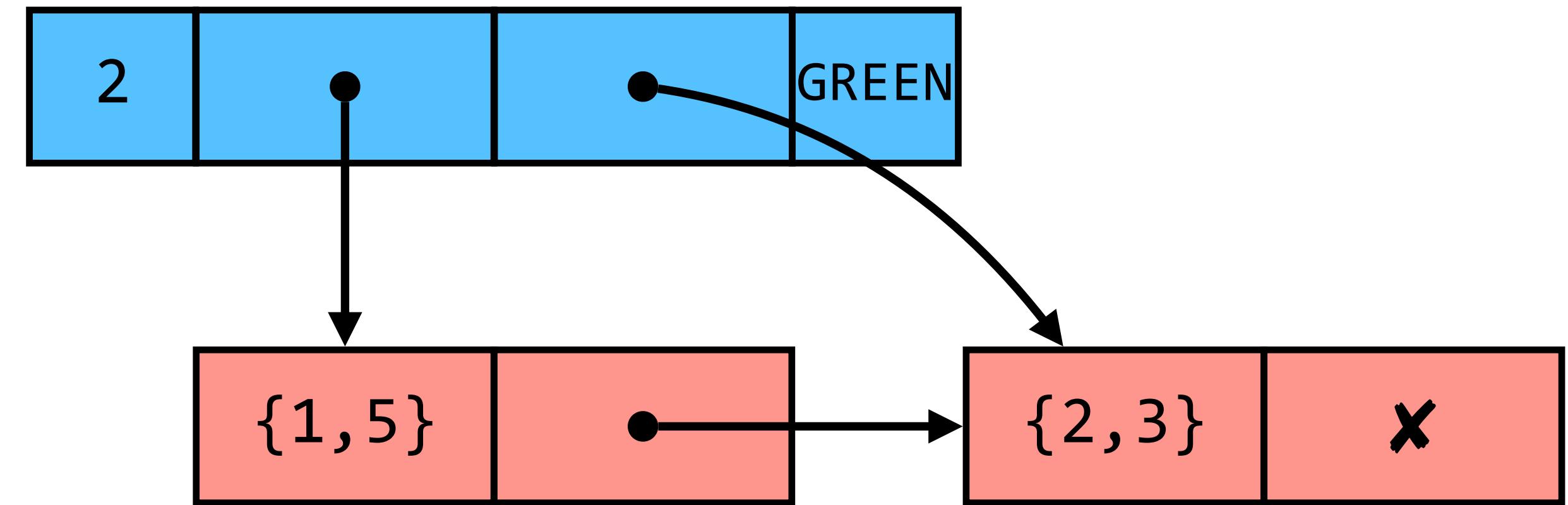
estende

```
int estende(poligonal *l, ponto p) {
    lponto new = malloc(sizeof(struct lponto_no));
    if (new == NULL) return 1;
    new->ponto = p;
    new->prox = NULL;
    l->tamanho++;
    if (l->pontos == NULL) l->pontos = new;
    else {
        lponto aux;
        for (aux = l->pontos; aux->prox != NULL; aux = aux->prox);
        aux->prox = new;
    }
    return 0;
}
```



Poligonal com lista ligada

```
typedef struct l ponto_no {  
    ponto ponto;  
    struct l ponto_no *prox;  
} *l ponto;  
  
typedef struct {  
    int tamanho;  
    l ponto pontos;  
    l ponto ultimo;  
    cor cor;  
} poligonal;
```



estende

```
int estende(poligonal *l, ponto p) {
    l ponto new = malloc(sizeof(struct lponto_no));
    if (new == NULL) return 1;
    new->ponto = p;
    new->prox = NULL;
    l->tamanho++;
    if (l->pontos == NULL) l->pontos = new;
    else l->ultimo->prox = new;
    l->ultimo = new;
    return 0;
}
```

aloca e liberta

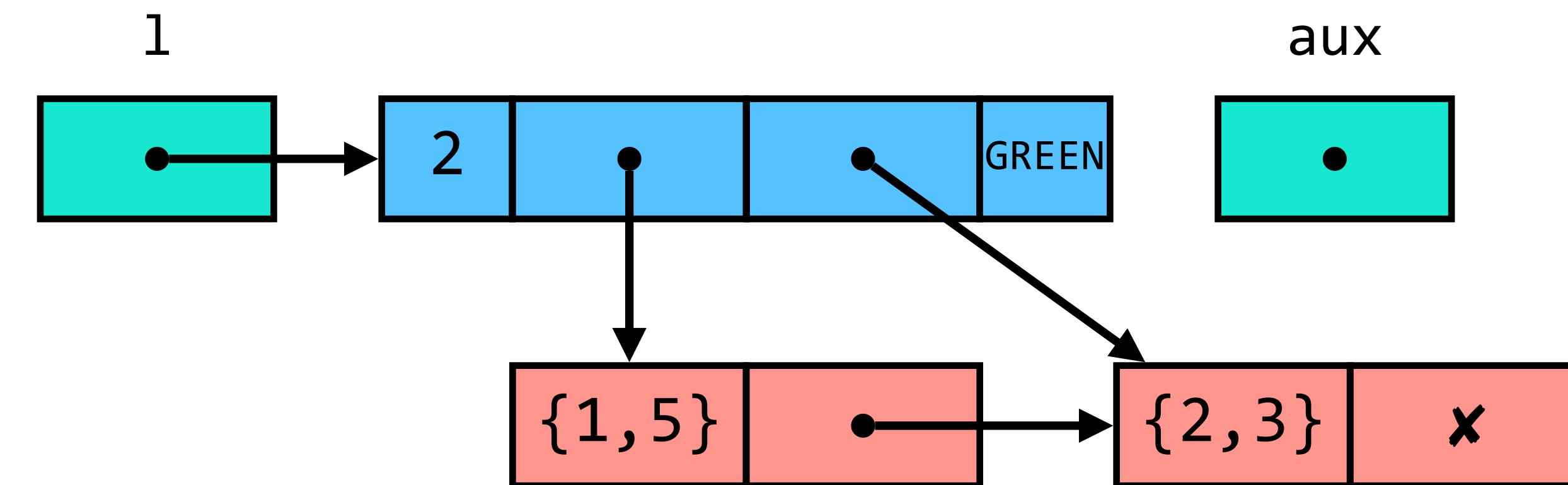
```
int aloca(poligonal *l) {
    l->tamanho = 0;
    l->pontos = NULL;
    l->ultimo = NULL;
    return 0;
}

void liberta(poligonal *l) {
    lponto aux;
    while (l->pontos != NULL) {
        aux = l->pontos;
        l->pontos = l->pontos->prox;
        free(aux);
    }
}
```

aloca e liberta

```
int aloca(poligonal *l) {  
    l->tamanho = 0;  
    l->pontos = NULL;  
    l->ultimo = NULL;  
    return 0;  
}
```

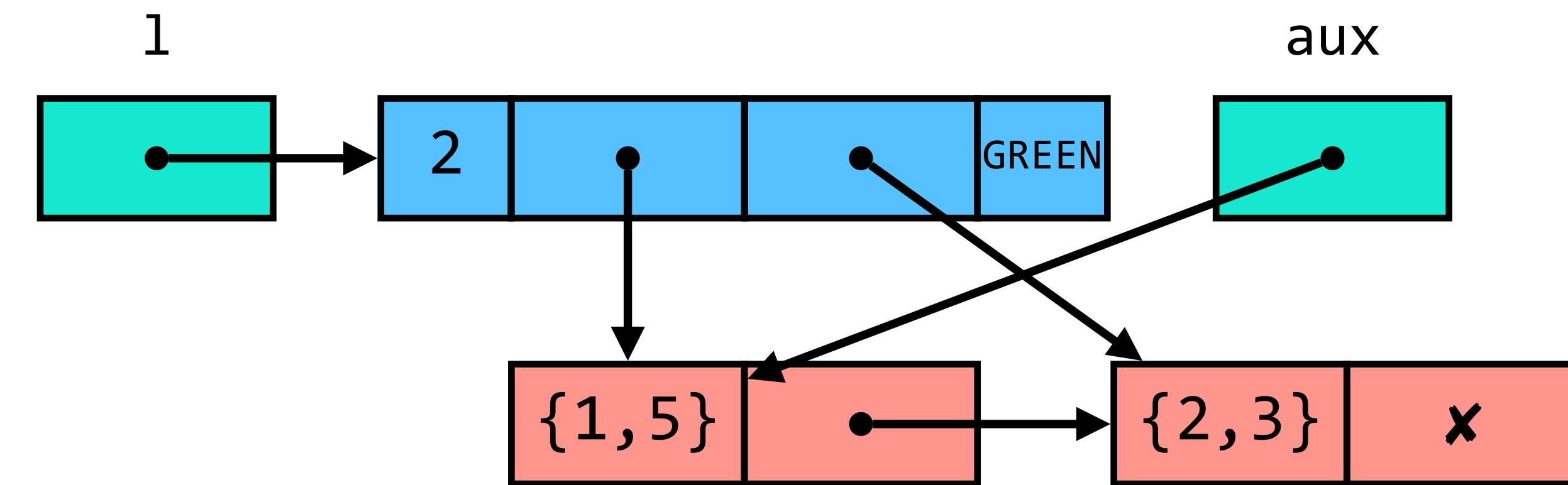
```
void liberta(poligonal *l) {  
    lponto aux;  
    while (l->pontos != NULL) {  
        aux = l->pontos;  
        l->pontos = l->pontos->prox;  
        free(aux);  
    }  
}
```



aloca e liberta

```
int aloca(poligonal *l) {  
    l->tamanho = 0;  
    l->pontos = NULL;  
    l->ultimo = NULL;  
    return 0;  
}
```

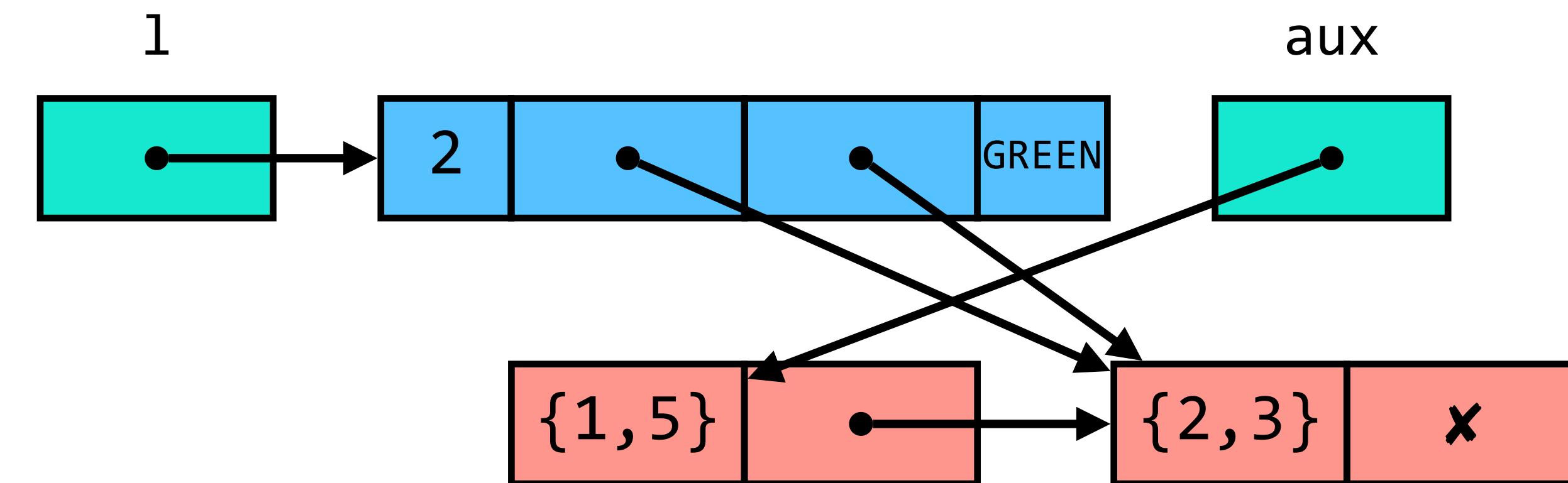
```
void liberta(poligonal *l) {  
    lponto aux;  
    while (l->pontos != NULL) {  
        aux = l->pontos;  
        l->pontos = l->pontos->prox;  
        free(aux);  
    }  
}
```



aloca e liberta

```
int aloca(poligonal *l) {  
    l->tamanho = 0;  
    l->pontos = NULL;  
    l->ultimo = NULL;  
    return 0;  
}
```

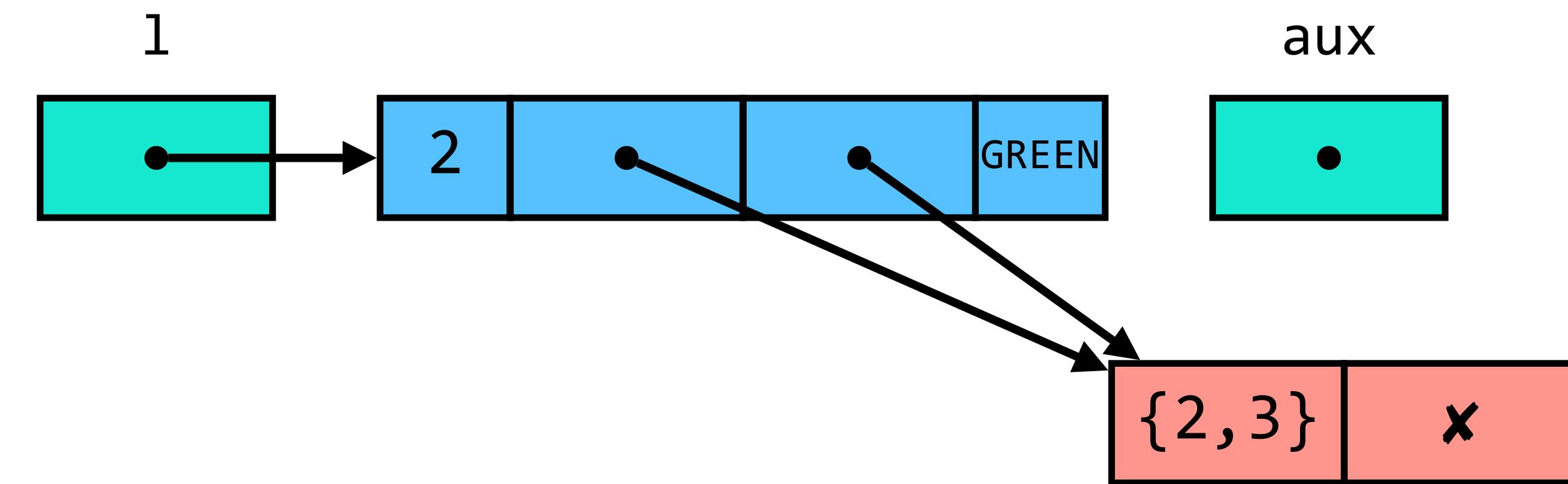
```
void liberta(poligonal *l) {  
    lponto aux;  
    while (l->pontos != NULL) {  
        aux = l->pontos;  
        l->pontos = l->pontos->prox;  
        free(aux);  
    }  
}
```



aloca e liberta

```
int aloca(poligonal *l) {  
    l->tamanho = 0;  
    l->pontos = NULL;  
    l->ultimo = NULL;  
    return 0;  
}
```

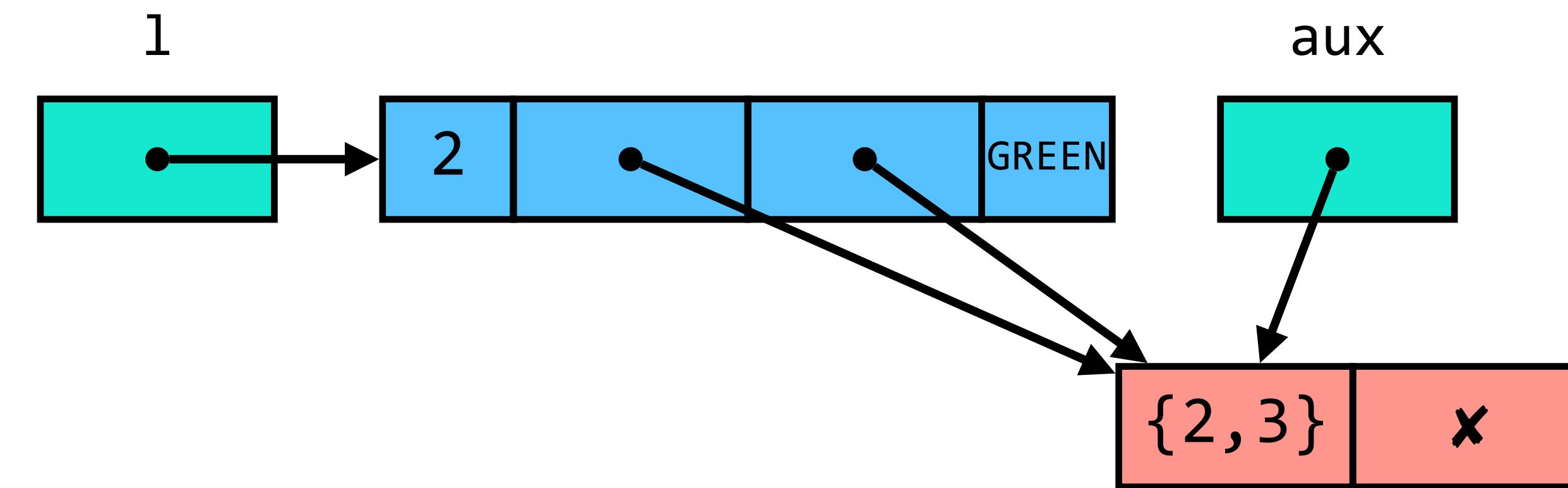
```
void liberta(poligonal *l) {  
    lponto aux;  
    while (l->pontos != NULL) {  
        aux = l->pontos;  
        l->pontos = l->pontos->prox;  
        free(aux);  
    }  
}
```



aloca e liberta

```
int aloca(poligonal *l) {  
    l->tamanho = 0;  
    l->pontos = NULL;  
    l->ultimo = NULL;  
    return 0;  
}
```

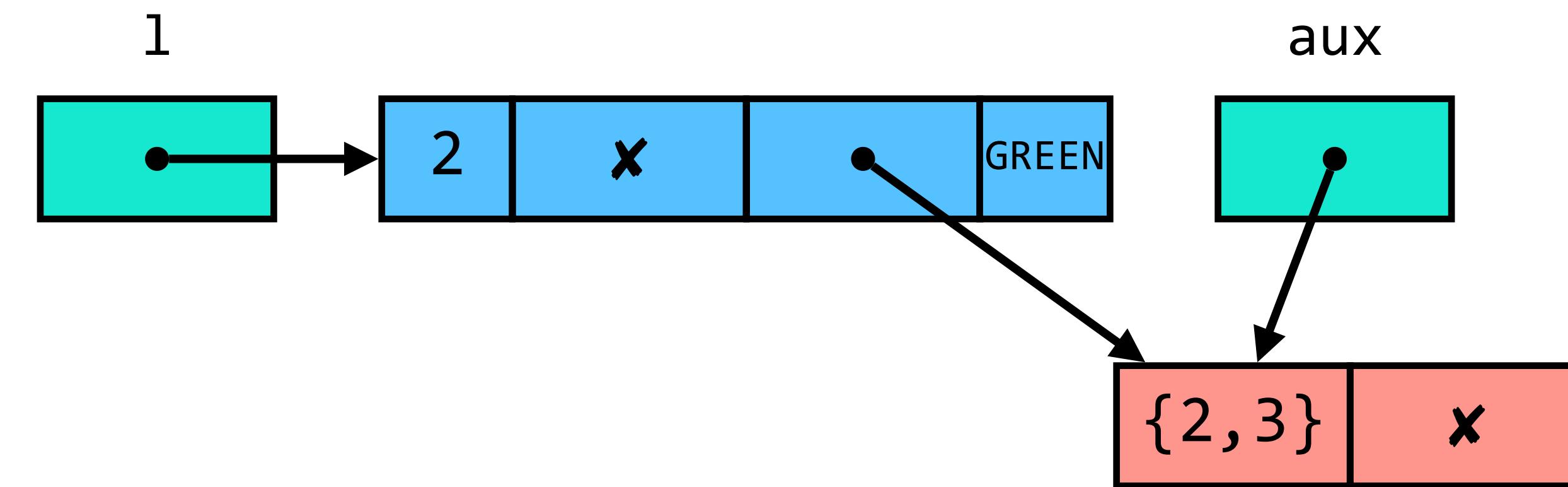
```
void liberta(poligonal *l) {  
    lponto aux;  
    while (l->pontos != NULL) {  
        aux = l->pontos;  
        l->pontos = l->pontos->prox;  
        free(aux);  
    }  
}
```



aloca e liberta

```
int aloca(poligonal *l) {  
    l->tamanho = 0;  
    l->pontos = NULL;  
    l->ultimo = NULL;  
    return 0;  
}
```

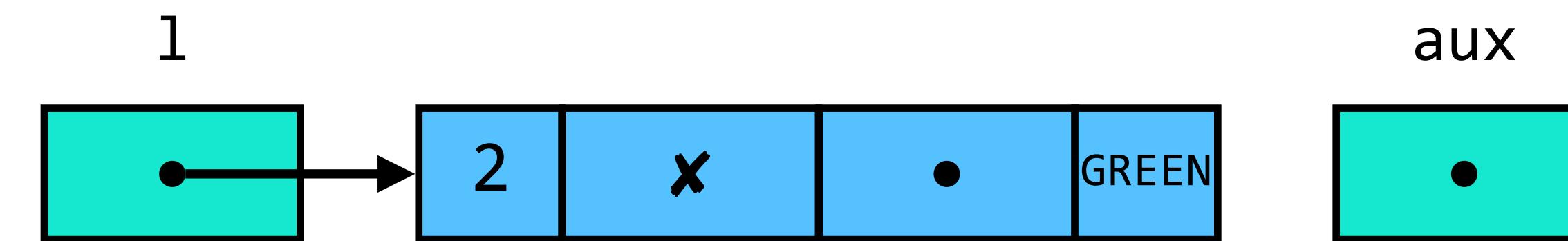
```
void liberta(poligonal *l) {  
    lponto aux;  
    while (l->pontos != NULL) {  
        aux = l->pontos;  
        l->pontos = l->pontos->prox;  
        free(aux);  
    }  
}
```



aloca e liberta

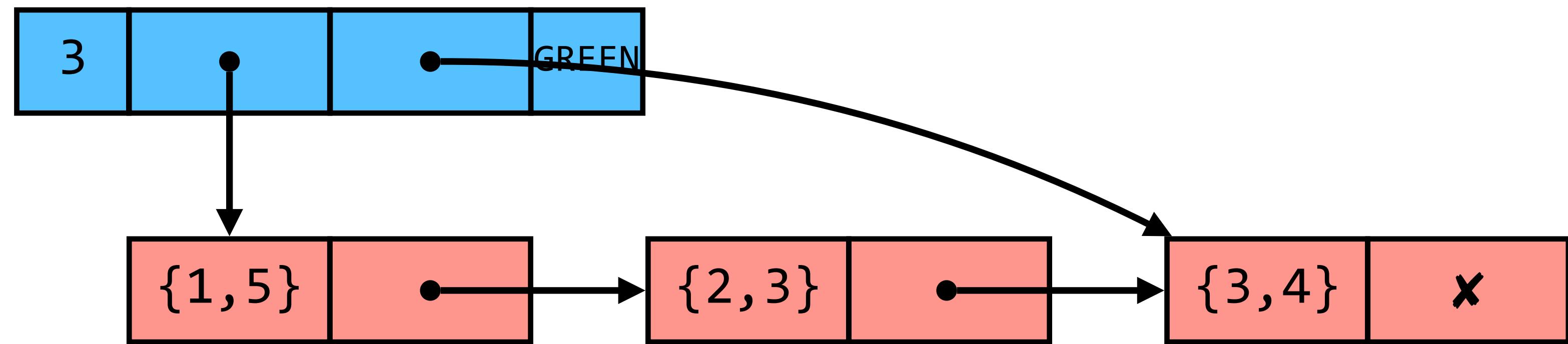
```
int aloca(poligonal *l) {  
    l->tamanho = 0;  
    l->pontos = NULL;  
    l->ultimo = NULL;  
    return 0;  
}
```

```
void liberta(poligonal *l) {  
    lponto aux;  
    while (l->pontos != NULL) {  
        aux = l->pontos;  
        l->pontos = l->pontos->prox;  
        free(aux);  
    }  
}
```



Poligonal com lista ligada

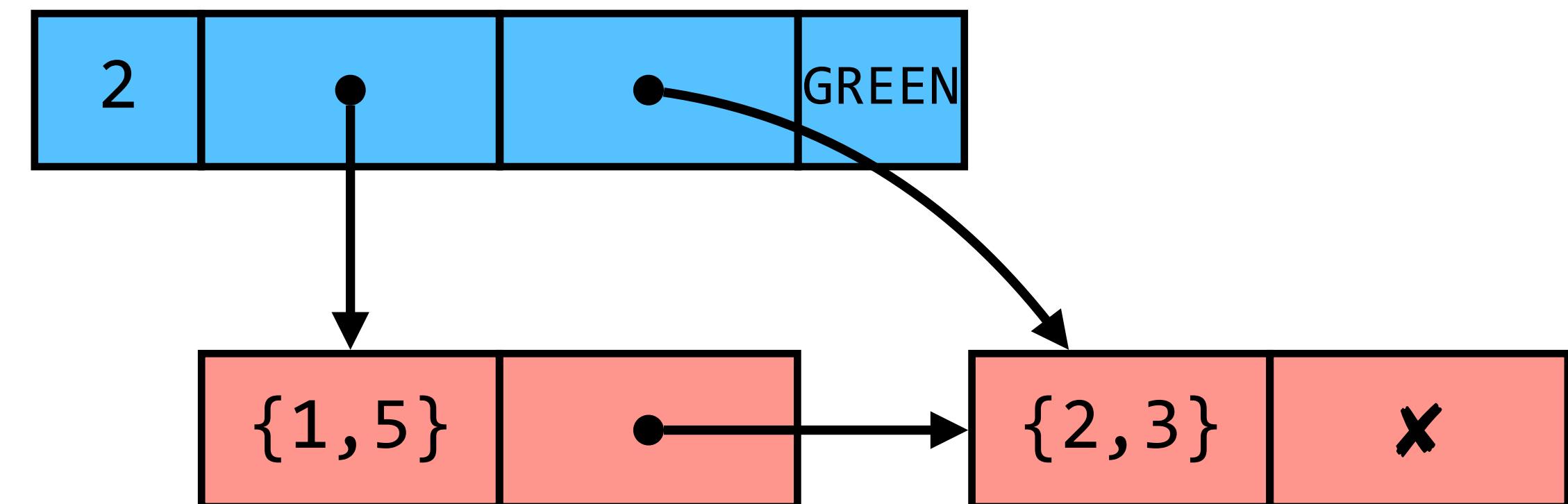
```
int main() {
    ponto a = {1,5}, b = {2,3}, c = {3,4};
    poligonal l;
    aloca(&l);
    l.cor = GREEN;
    estende(&l,a);
    estende(&l,b);
    estende(&l,c);
    printf("%.1f",comprimento(&l));
    liberta(&l);
    return 0;
}
```



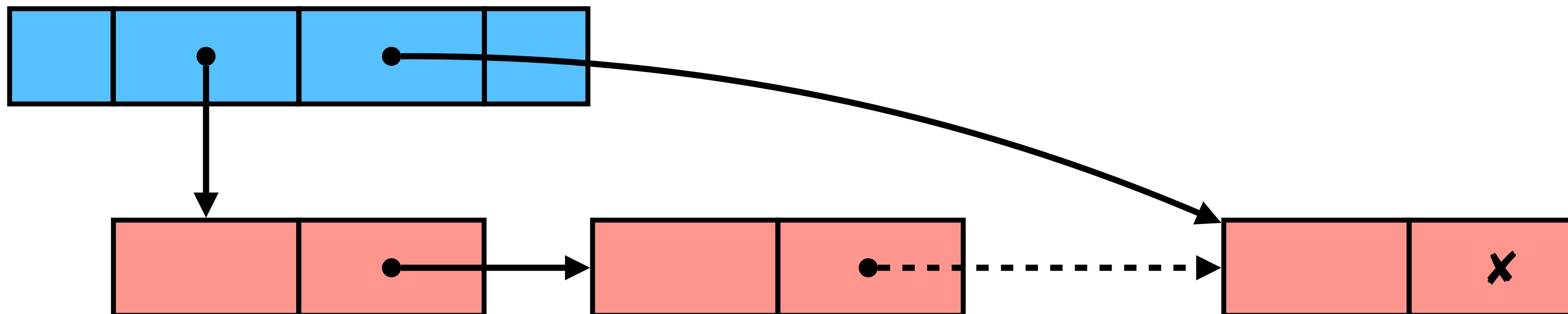
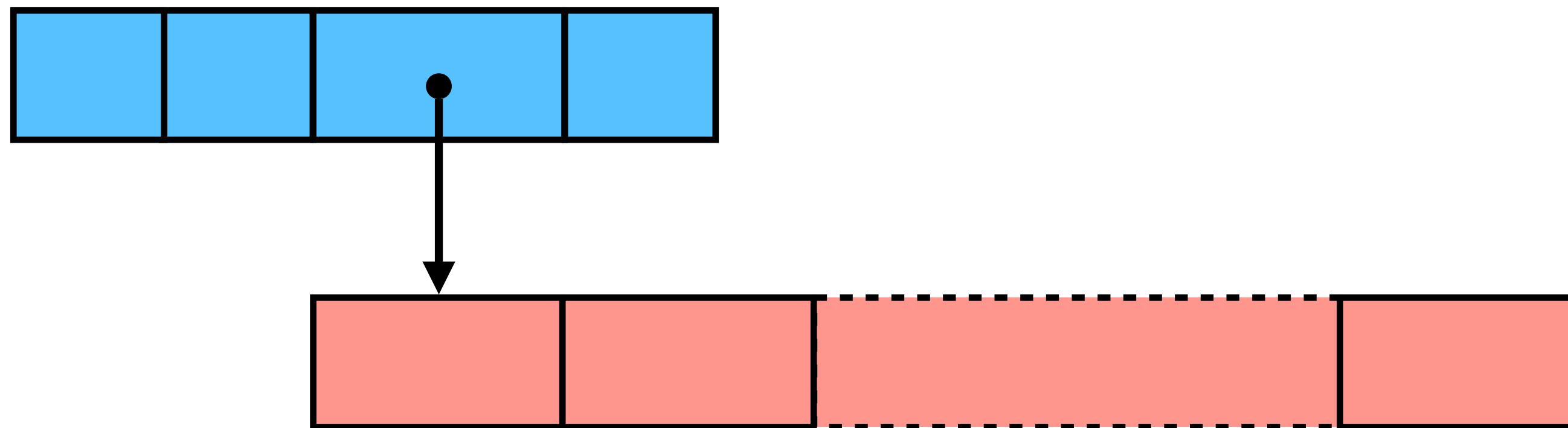
#20 É verdade que `tamanho(&l) == tamanho(&l)`?

```
int tamanho(poligonal *l) {  
    int r = 0;  
    while (l->pontos != NULL) {  
        l->pontos = l->pontos->prox;  
        r++;  
    }  
    return r;  
}
```

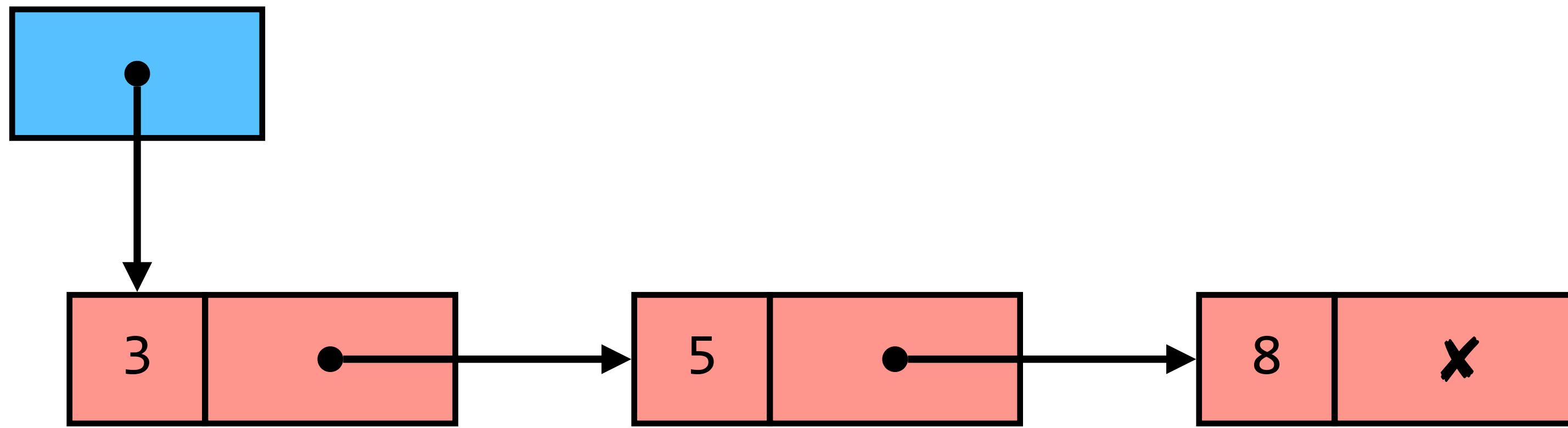
```
int main() {  
    poligonal l = ...;  
    printf("%d\n", tamanho(&l) == tamanho(&l));  
    return 0;  
}
```



#21 Qual ocupa menos memória?



Listas ligadas de inteiros

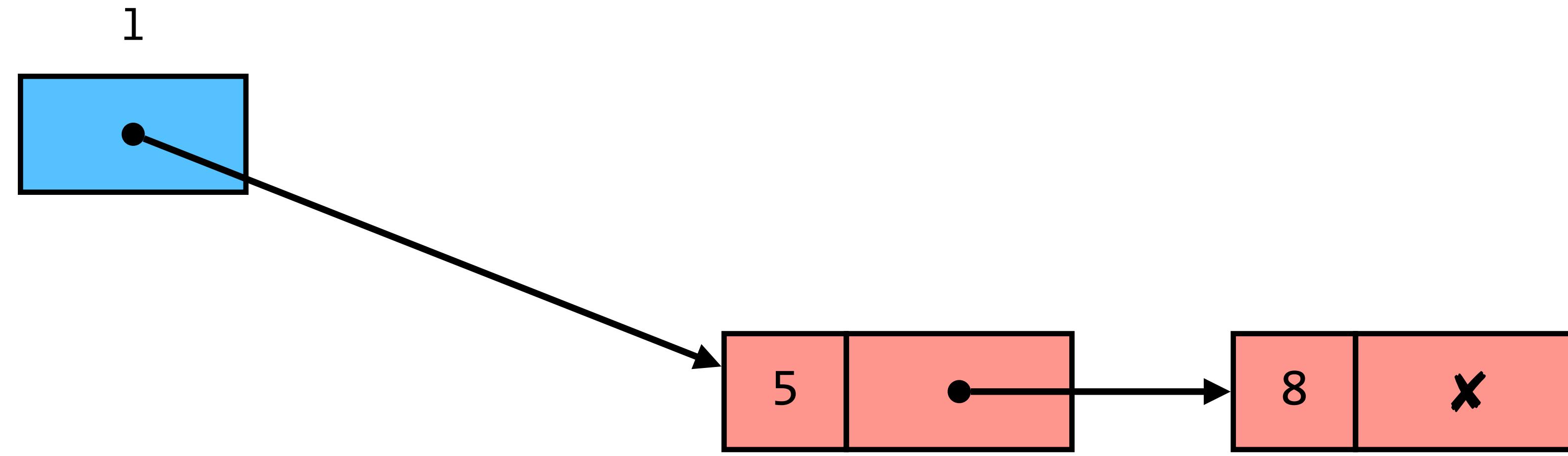


```
typedef struct lint_no {  
    int valor;  
    struct lint_no *prox;  
} *lint;
```

Inserção à cabeça

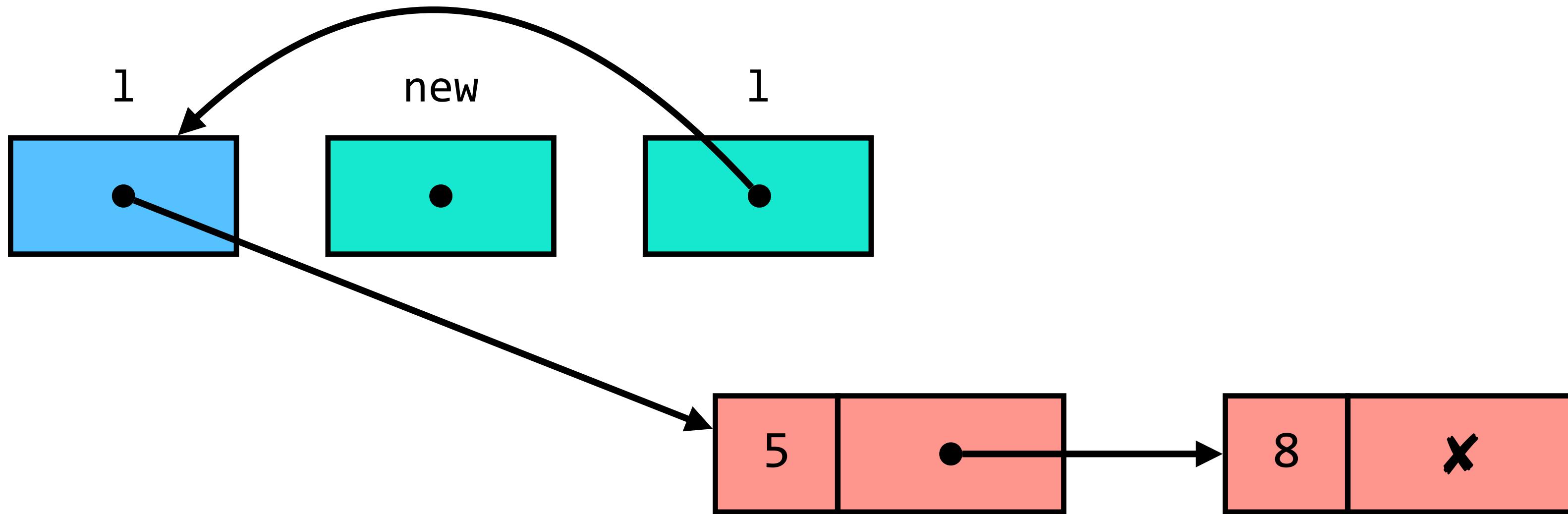
```
int cons(int x, lint *l) {  
    lint new = malloc(sizeof(struct lint_no));  
    if (new == NULL) return 1;  
    new->valor = x;  
    new->prox = l;  
    *l = new;  
    return 0;  
}
```

```
int main() {  
    lint l = NULL;  
    cons(8,&l);  
    cons(5,&l);  
    cons(3,&l);  
    return 0;  
}
```



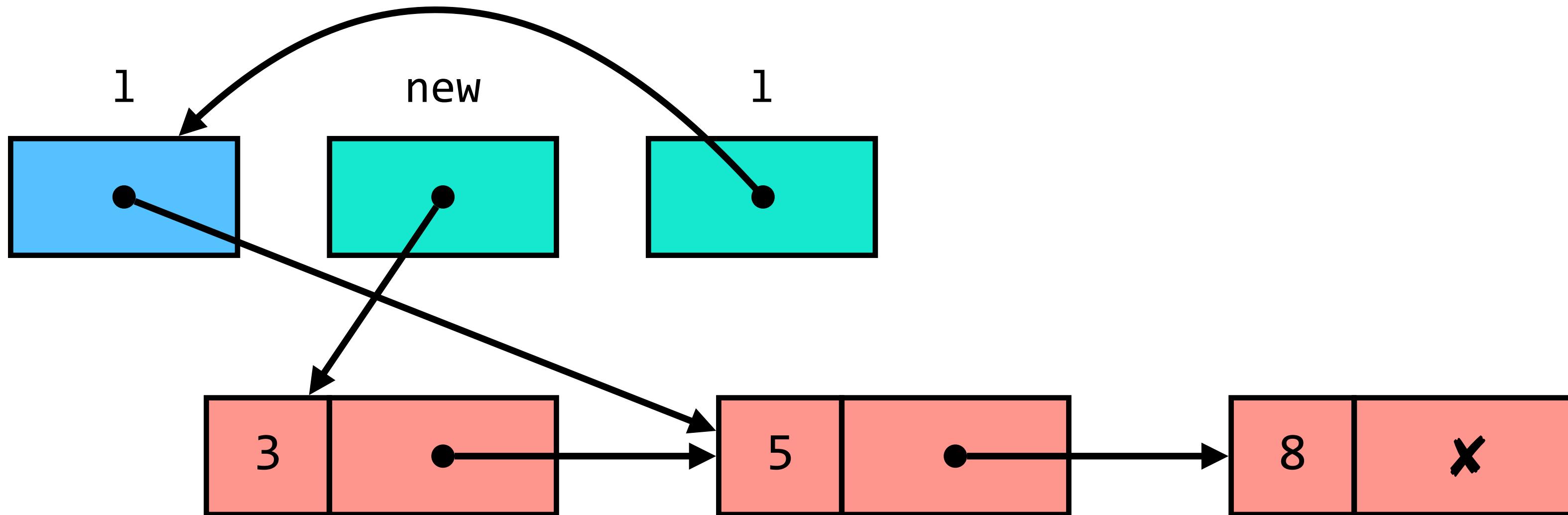
Inserção à cabeça

```
int cons(int x, lint *l) {  
    lint new = malloc(sizeof(struct lint_no));  
    if (new == NULL) return 1;  
    new->valor = x;  
    new->prox = *l;  
    *l = new;  
    return 0;  
}  
  
int main() {  
    lint l = NULL;  
    cons(8,&l);  
    cons(5,&l);  
    cons(3,&l);  
    return 0;  
}
```



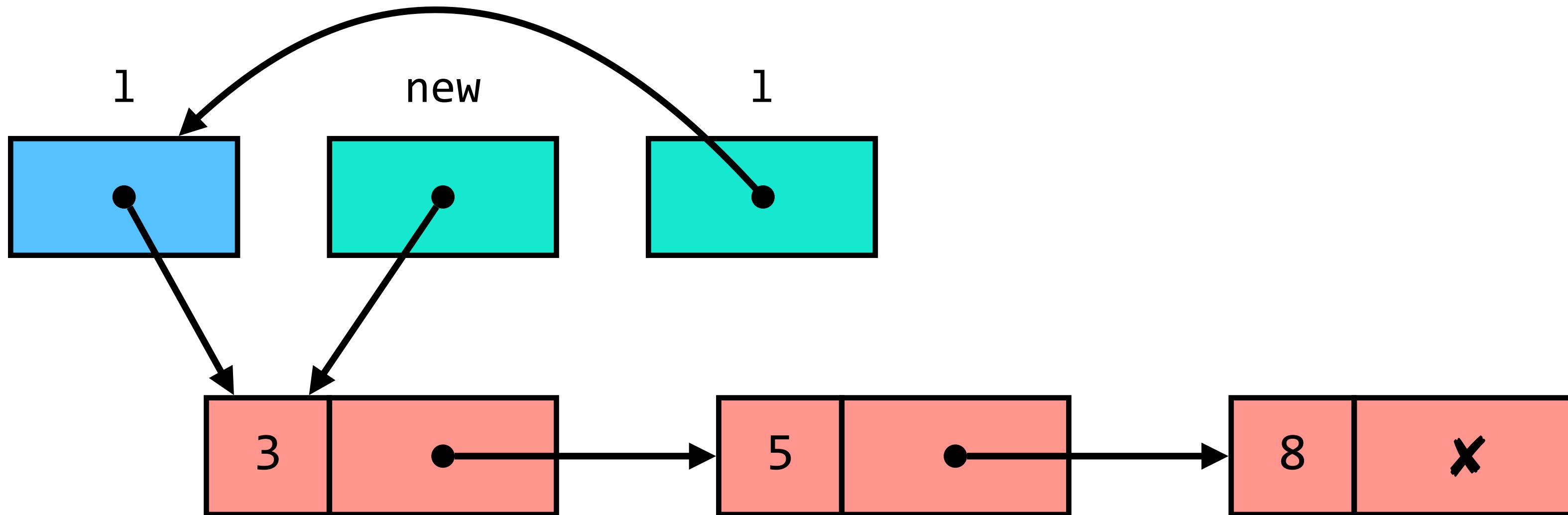
Inserção à cabeça

```
int cons(int x, lint *l) {  
    lint new = malloc(sizeof(struct lint_no));  
    if (new == NULL) return 1;  
    new->valor = x;  
    new->prox = *l;  
    *l = new;  
    return 0;  
}  
  
int main() {  
    lint l = NULL;  
    cons(8,&l);  
    cons(5,&l);  
    cons(3,&l);  
    return 0;  
}
```



Inserção à cabeça

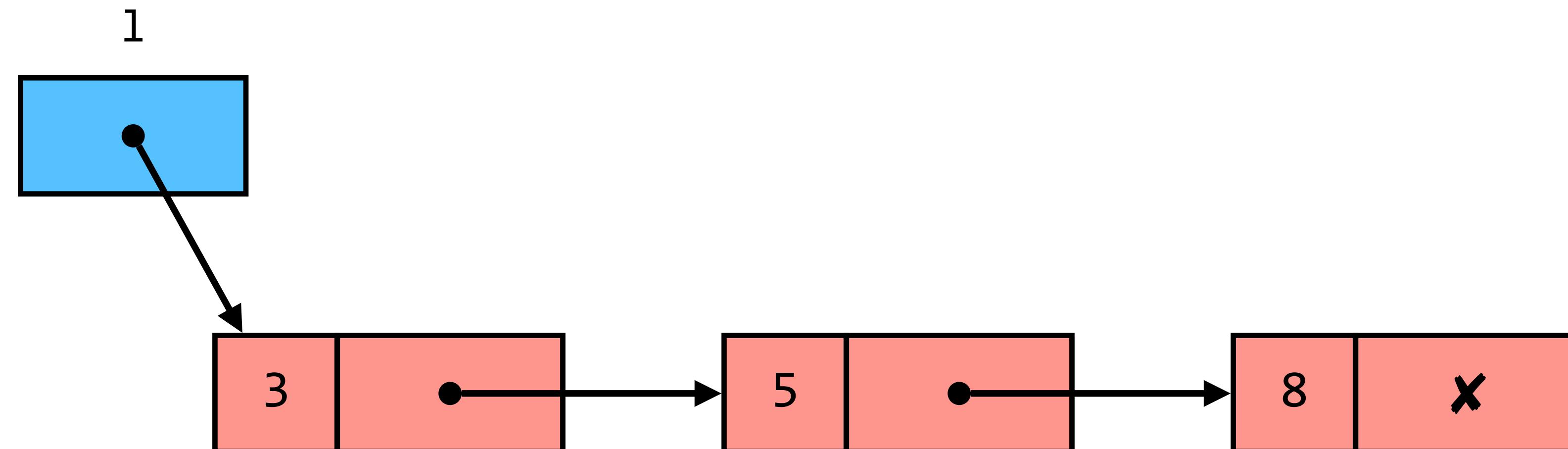
```
int cons(int x, lint *l) {  
    lint new = malloc(sizeof(struct lint_no));  
    if (new == NULL) return 1;  
    new->valor = x;  
    new->prox = *l;  
    *l = new;  
    return 0;  
}  
  
int main() {  
    lint l = NULL;  
    cons(8,&l);  
    cons(5,&l);  
    cons(3,&l);  
    return 0;  
}
```



Inserção à cabeça

```
int cons(int x, lint *l) {  
    lint new = malloc(sizeof(struct lint_no));  
    if (new == NULL) return 1;  
    new->valor = x;  
    new->prox = *l;  
    *l = new;  
    return 0;  
}
```

```
int main() {  
    lint l = NULL;  
    cons(8,&l);  
    cons(5,&l);  
    cons(3,&l);  
    return 0;  
}
```



Aula 12

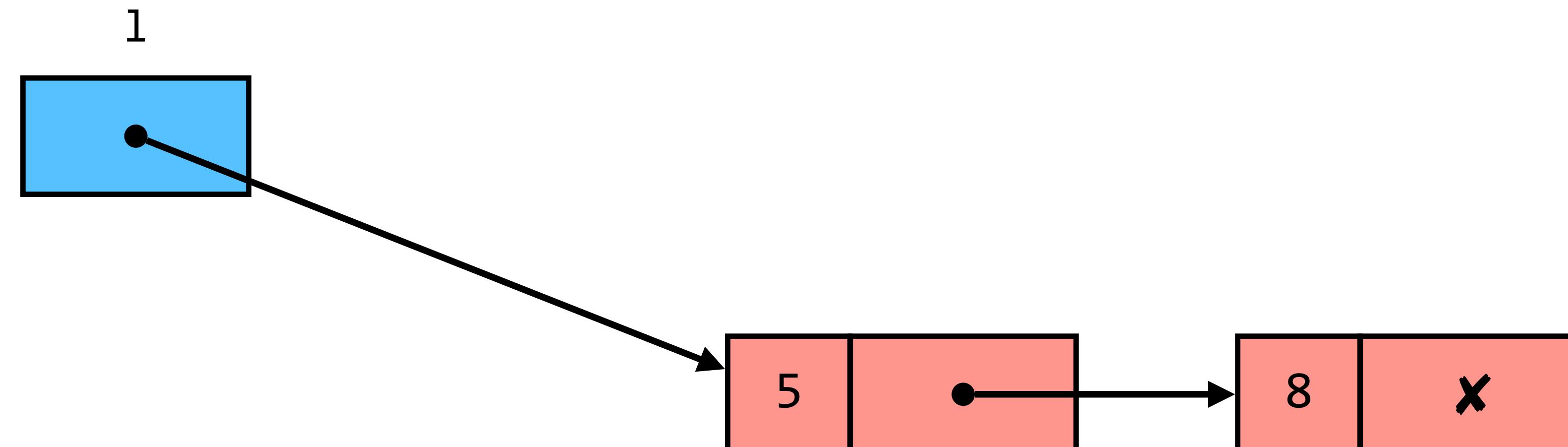
Disclaimer



Inserção à cabeça

```
lint cons(int x, lint l) {  
    lint new = malloc(sizeof(struct lint_no));  
    new->valor = x;  
    new->prox = l;  
    return new;  
}
```

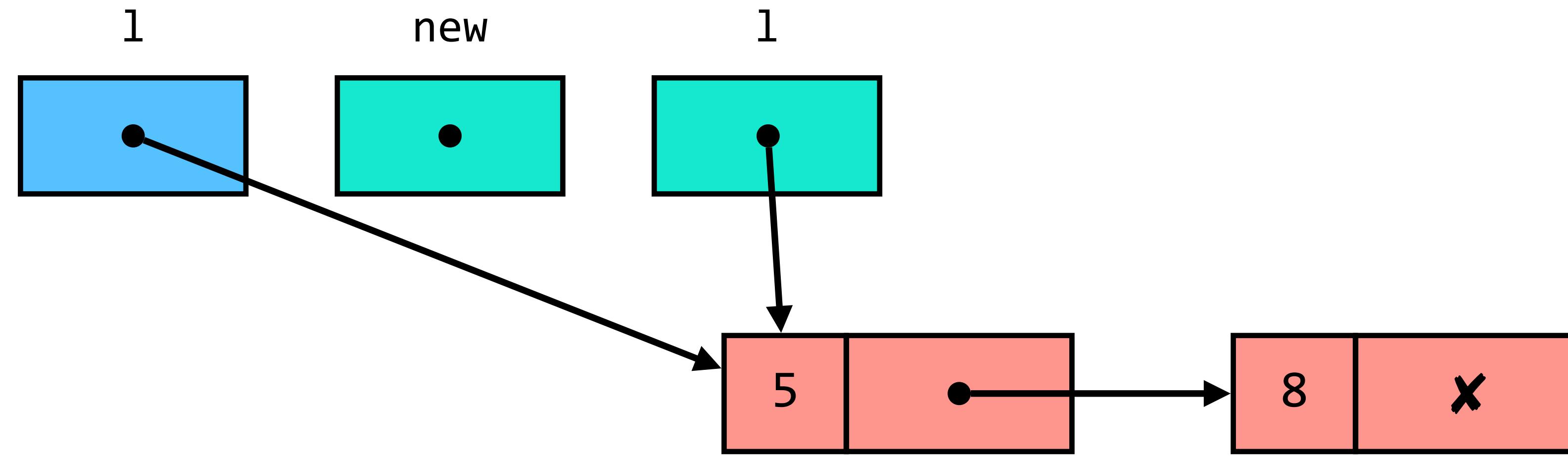
```
int main() {  
    lint l = NULL;  
    l = cons(8,l);  
    l = cons(5,l);  
    l = cons(3,l);  
    return 0;  
}
```



Inserção à cabeça

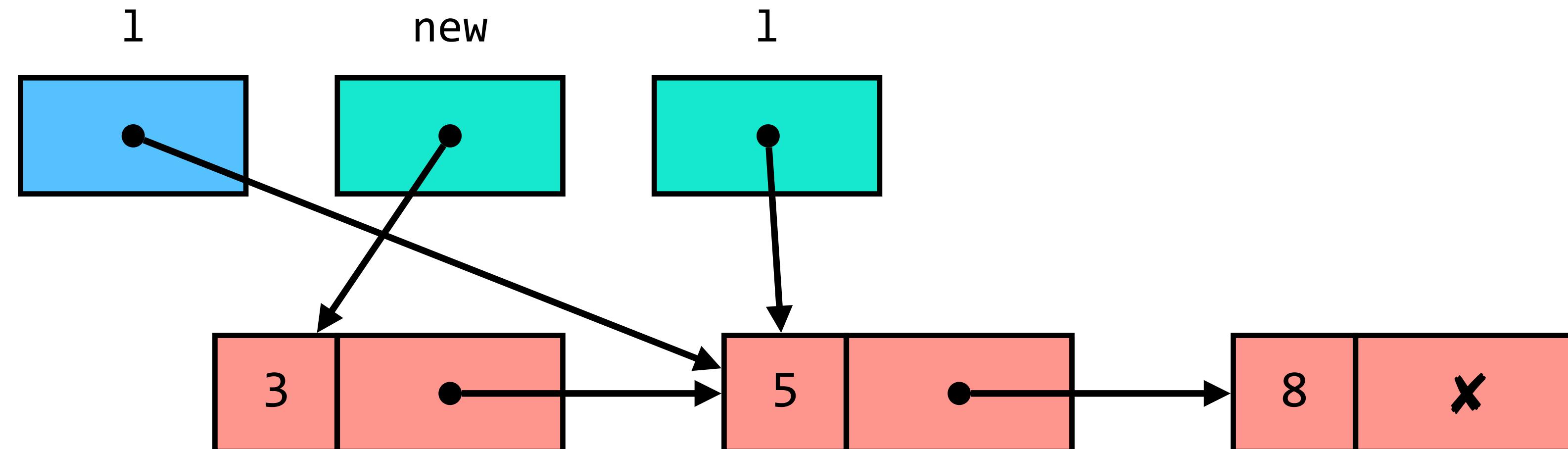
```
lint cons(int x, lint l) {  
    lint new = malloc(sizeof(struct lint_no));  
    new->valor = x;  
    new->prox = l;  
    return new;  
}
```

```
int main() {  
    lint l = NULL;  
    l = cons(8,l);  
    l = cons(5,l);  
    l = cons(3,l);  
    return 0;  
}
```



Inserção à cabeça

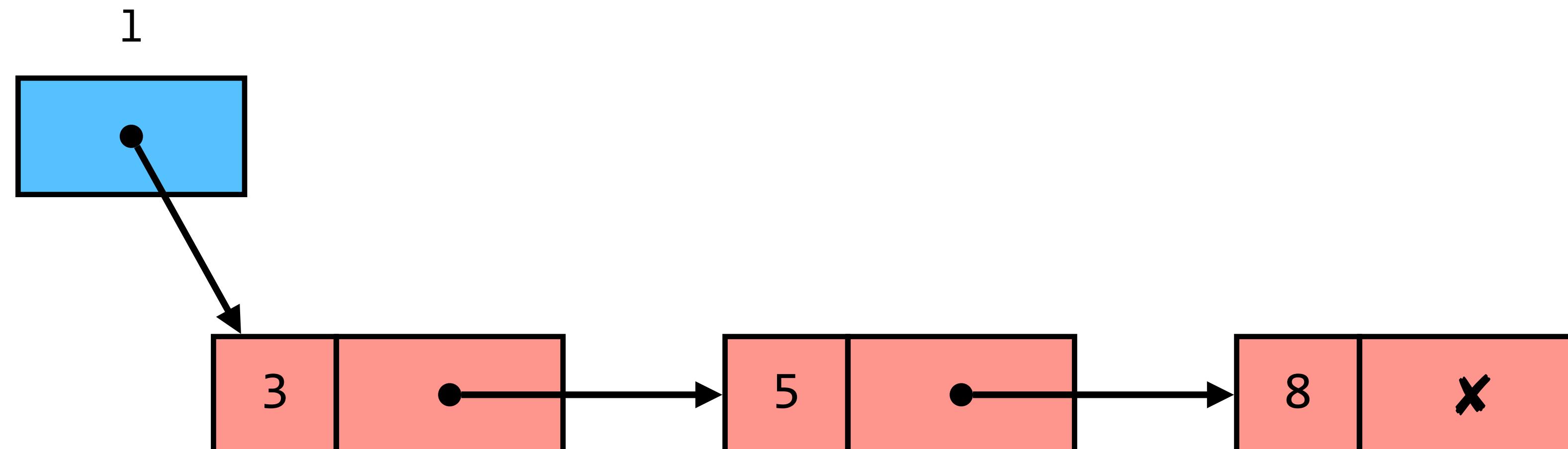
```
lint cons(int x, lint l) {  
    lint new = malloc(sizeof(struct lint_no));  
    new->valor = x;  
    new->prox = l;  
    return new;  
}  
  
int main() {  
    lint l = NULL;  
    l = cons(8,l);  
    l = cons(5,l);  
    l = cons(3,l);  
    return 0;  
}
```



Inserção à cabeça

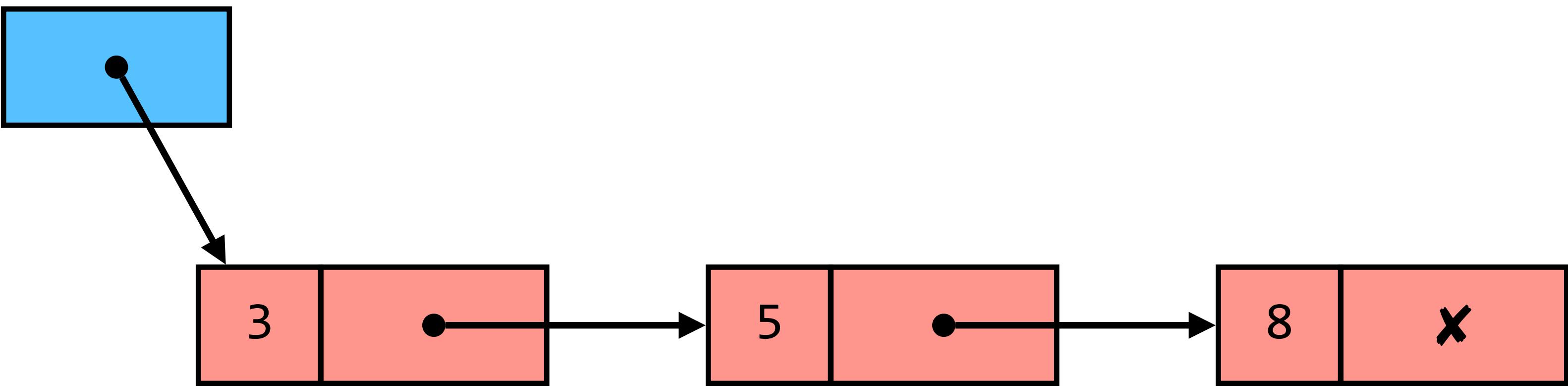
```
lint cons(int x, lint l) {  
    lint new = malloc(sizeof(struct lint_no));  
    new->valor = x;  
    new->prox = l;  
    return new;  
}
```

```
int main() {  
    lint l = NULL;  
    l = cons(8,l);  
    l = cons(5,l);  
    l = cons(3,l);  
    return 0;  
}
```



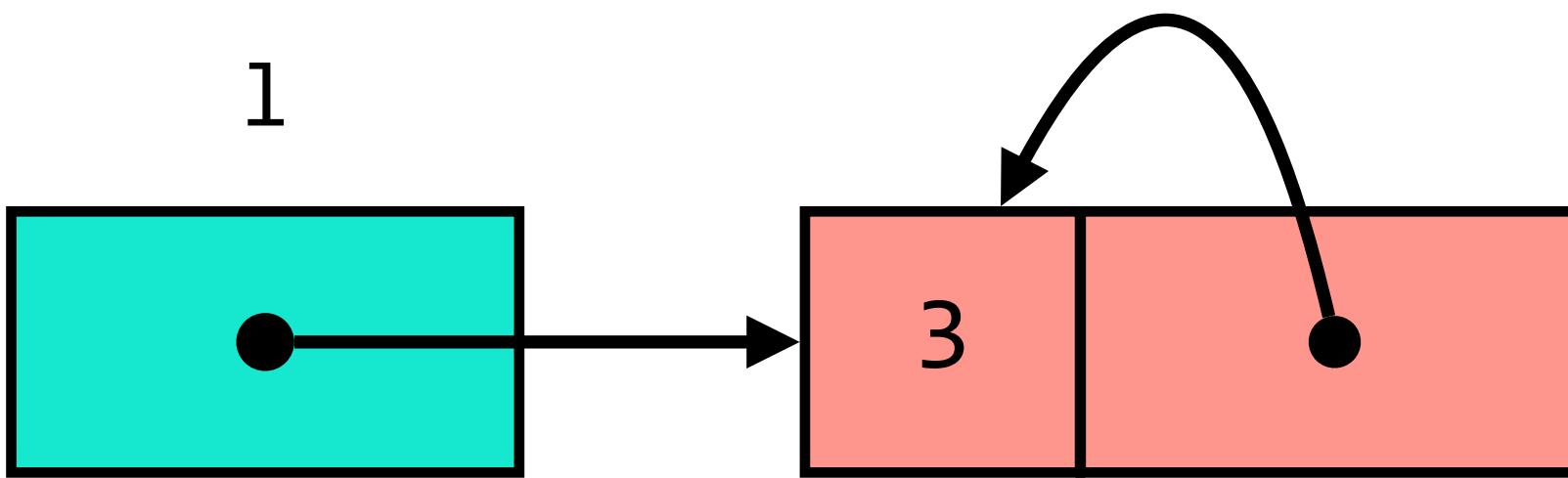
Inserção à cabeça

```
lint cons(int x, lint l) {  
    lint new = malloc(sizeof(struct lint_no));  
    new->valor = x;           1  
    new->prox = l;  
    return new;  
}  
  
int main() {  
    lint l = cons(3,cons(5,cons(8,NULL)));  
    return 0;  
}
```

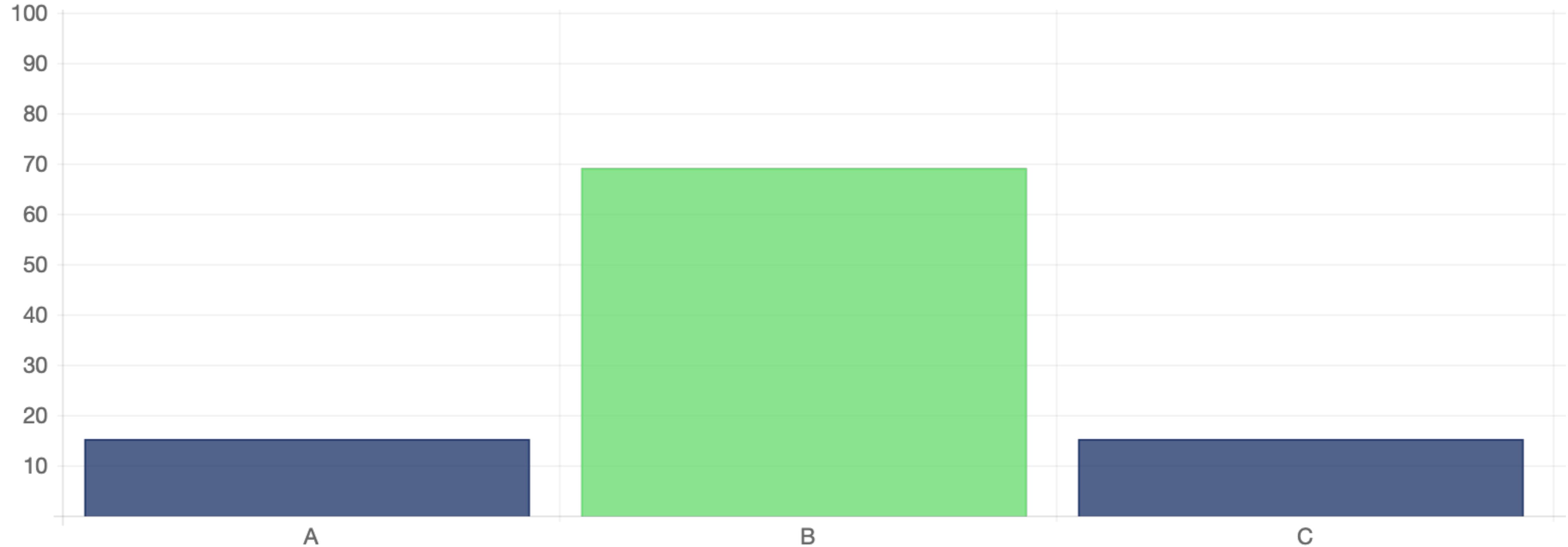


#22 Como criar esta “lista”?

```
lint A() {  
    lint l = cons(3, 1);  
    return l;  
}  
  
lint B() {  
    lint l = cons(3, NULL);  
    l->prox = l;  
    return l;  
}  
  
lint C() {  
    lint l = cons(3, NULL);  
    l = l->prox;  
    return l;  
}
```

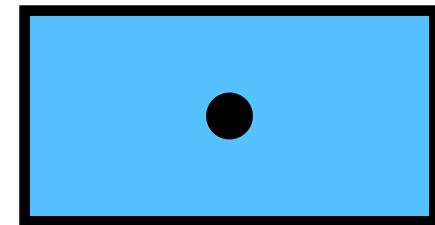


#22 Como criar esta “lista”?



Criação a partir de array

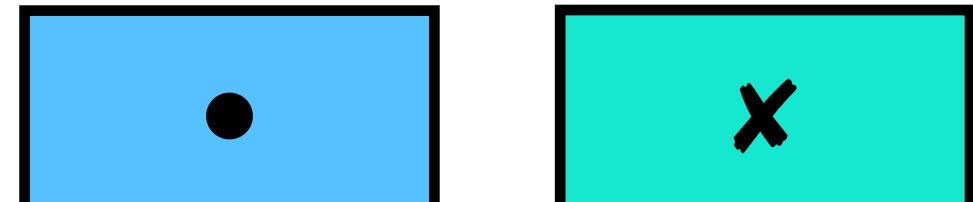
```
lint fromArray(int a[], int N) {  
    lint l = NULL;  
    for (int i = N-1; i >= 0; i--) l = cons(a[i], l);  
    return l;  
}
```



```
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v,3);  
    return 0;  
}
```

Criação a partir de array

```
lint fromArray(int a[], int N) {  
    lint l = NULL;  
    for (int i = N-1; i >= 0; i--) l = cons(a[i], l);  
    return l;  
}
```

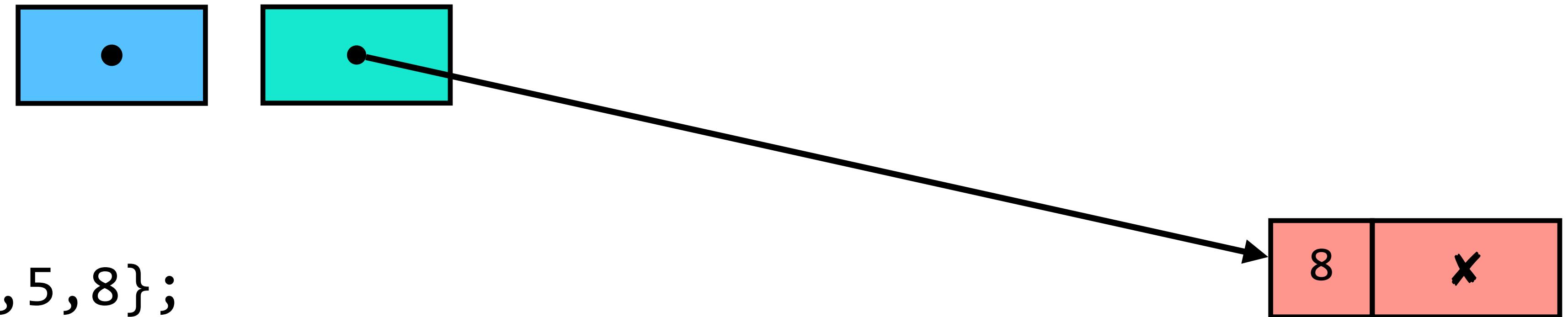


```
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v,3);  
    return 0;  
}
```

Criação a partir de array

```
lint fromArray(int a[], int N) {  
    lint l = NULL;  
    for (int i = N-1; i >= 0; i--) l = cons(a[i], l);  
    return l;  
}
```

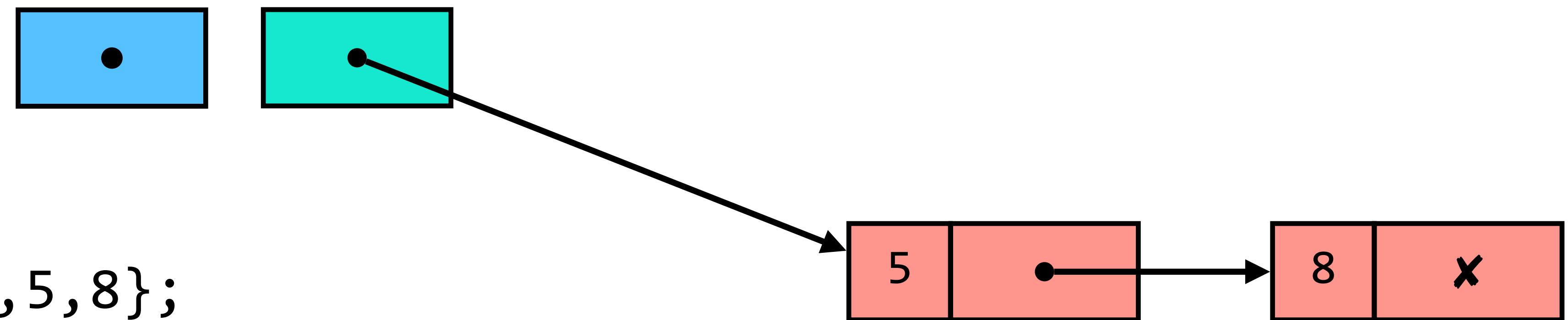
```
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v,3);  
    return 0;  
}
```



Criação a partir de array

```
lint fromArray(int a[], int N) {  
    lint l = NULL;  
    for (int i = N-1; i >= 0; i--) l = cons(a[i], l);  
    return l;  
}
```

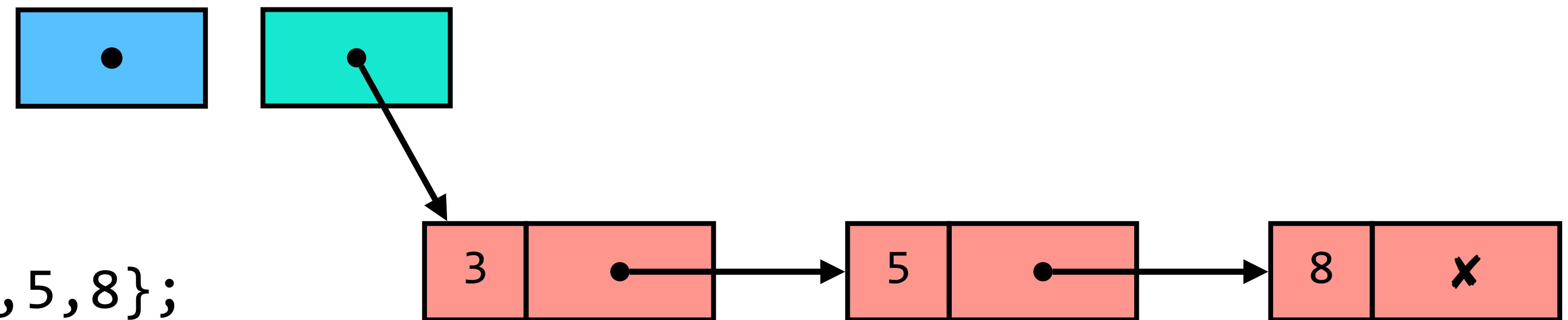
```
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v,3);  
    return 0;  
}
```



Criação a partir de array

```
lint fromArray(int a[], int N) {  
    lint l = NULL;  
    for (int i = N-1; i >= 0; i--) l = cons(a[i], l);  
    return l;  
}
```

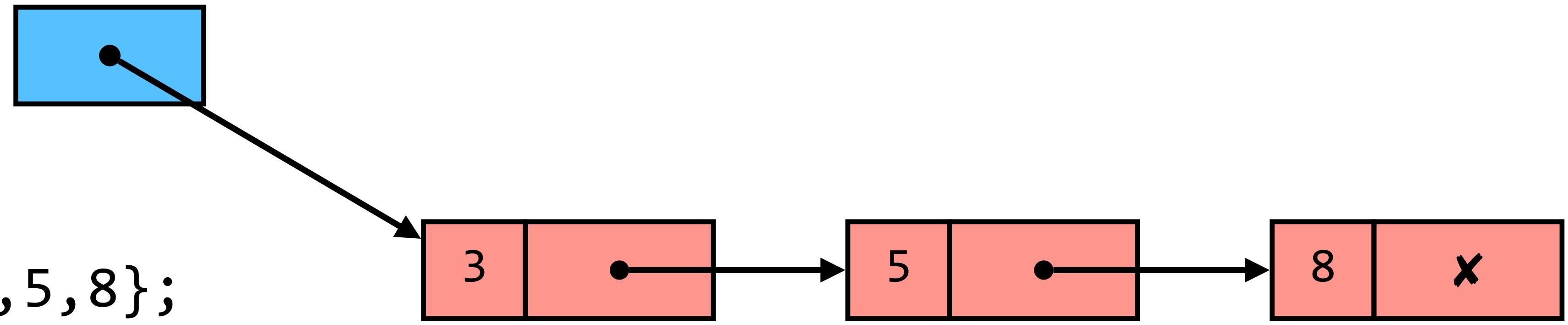
```
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v,3);  
    return 0;  
}
```



Criação a partir de array

```
lint fromArray(int a[], int N) {  
    lint l = NULL;  
    for (int i = N-1; i >= 0; i--) l = cons(a[i], l);  
    return l;  
}
```

```
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v,3);  
    return 0;  
}
```

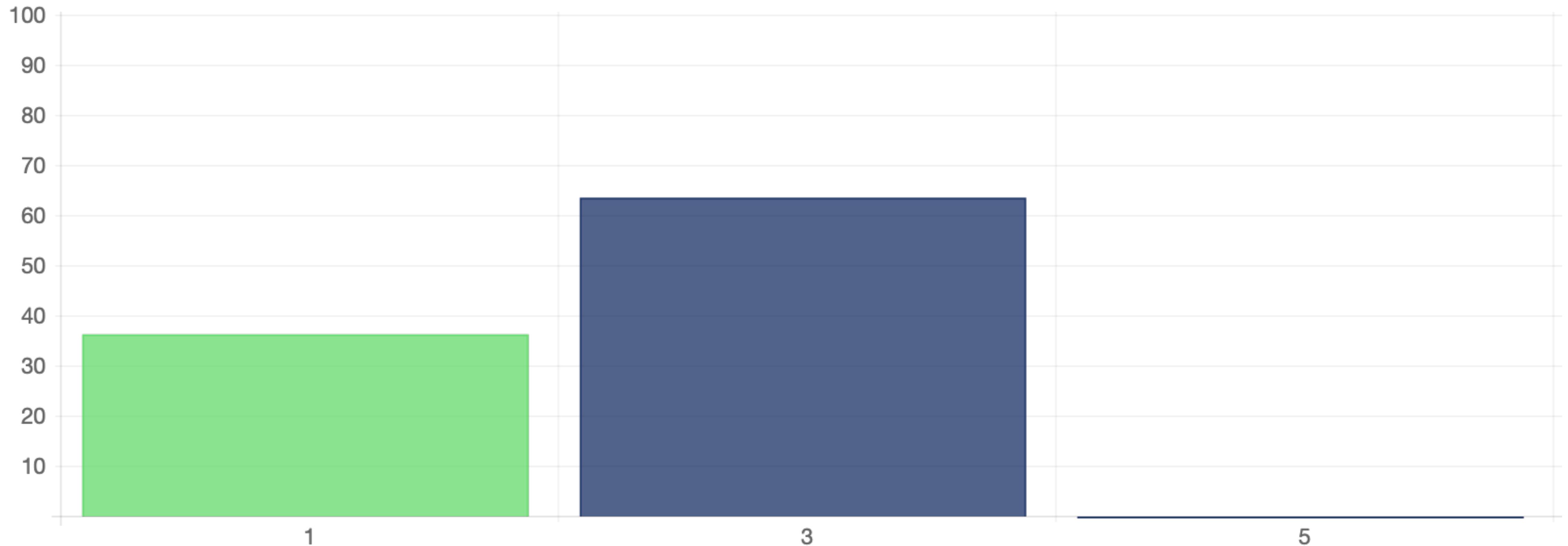


#23 Qual o valor na cabeça de 1?

```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v, 3);
    lint m = l;
    m->valor = 1;
    printf("%d\n", l->valor);
    return 0;
}
```



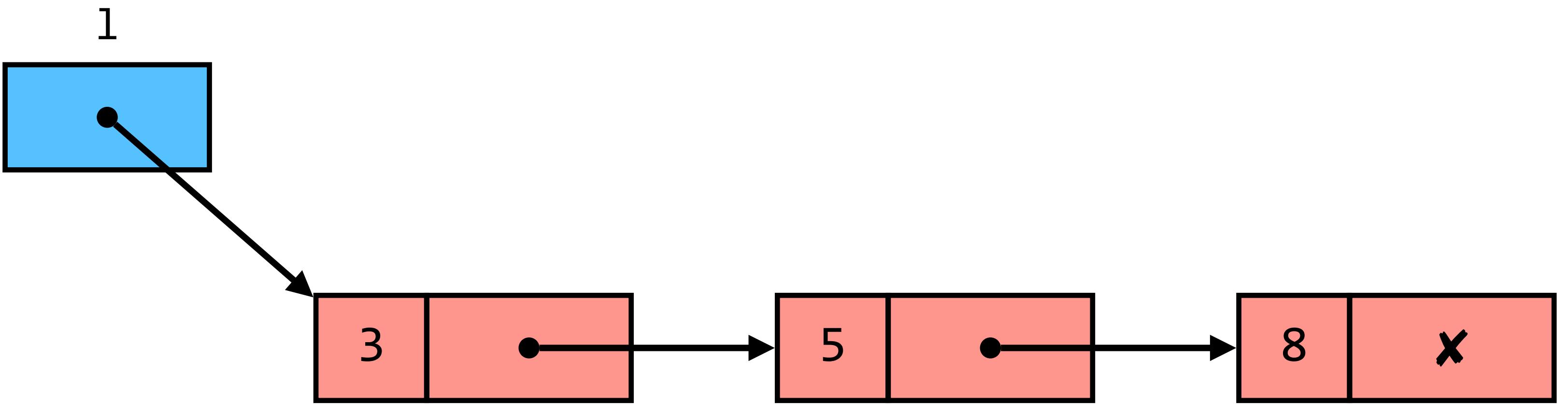
#23 Qual o valor na cabeça de 1?



Duplicação

```
lint clone(lint l) {  
    ...  
}
```

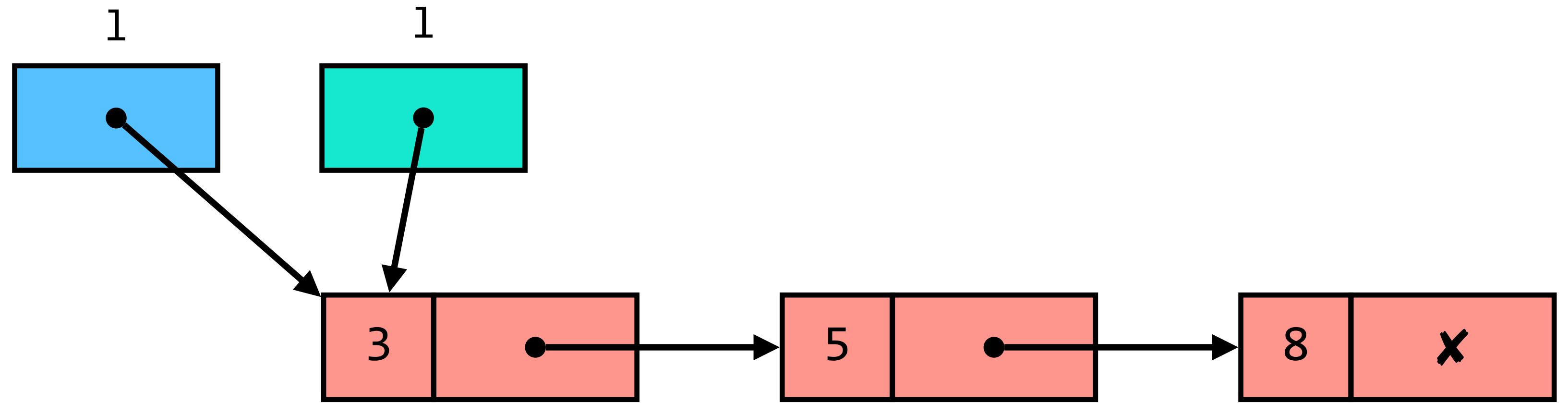
Duplicação



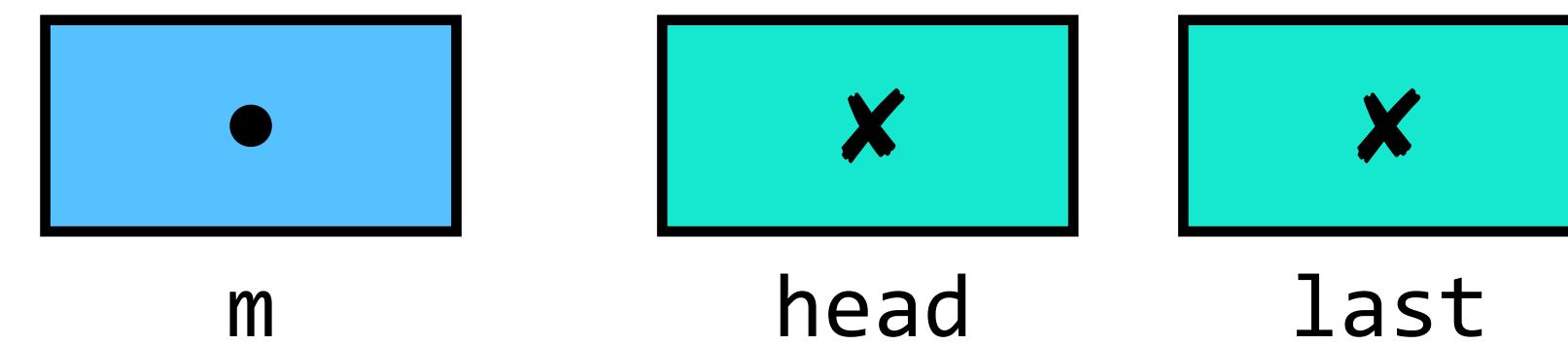
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    lint m = clone(l);
    return 0;
}
```

m

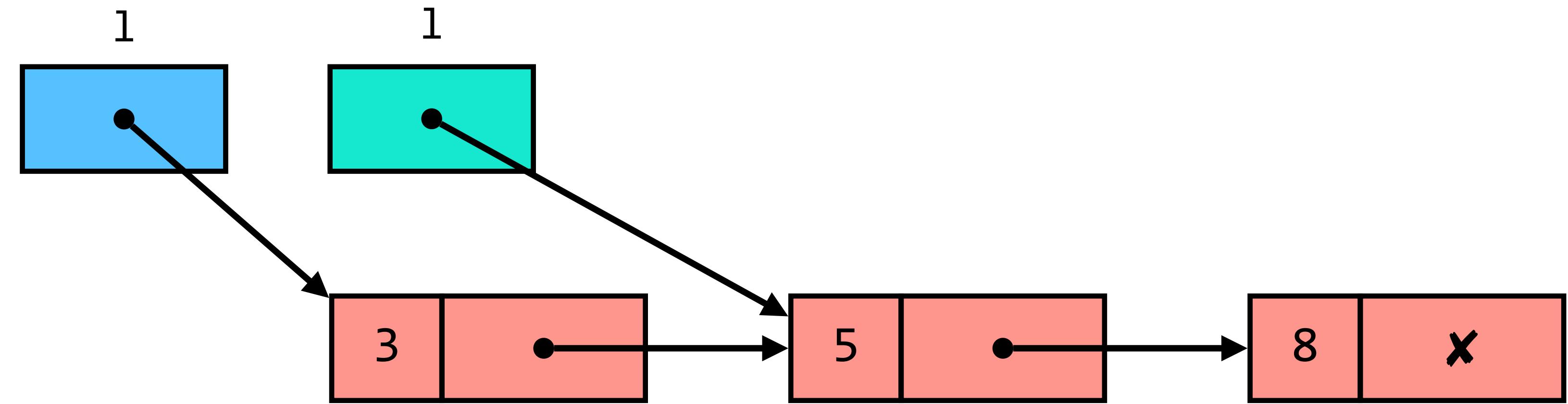
Duplicação



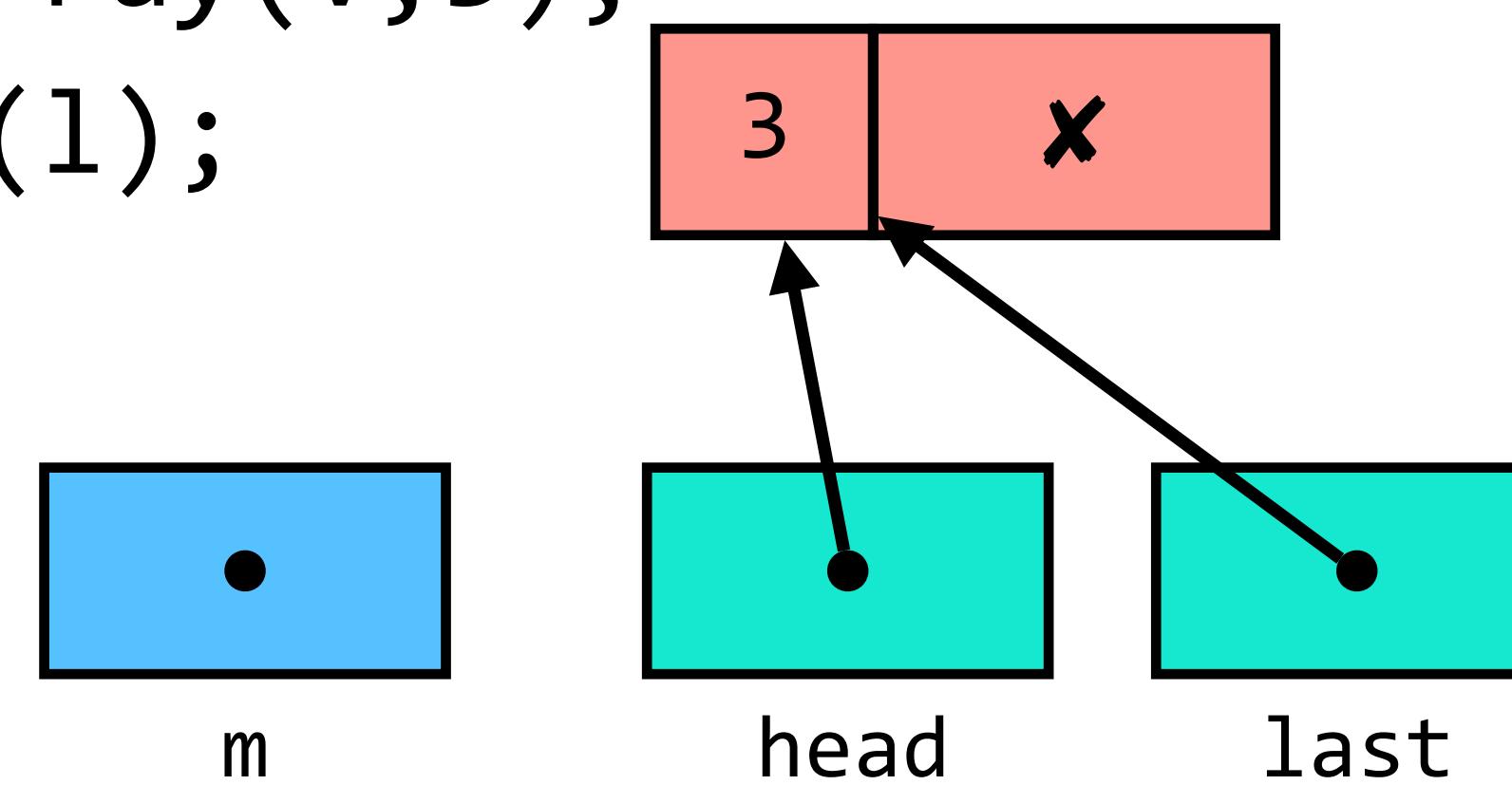
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    lint m = clone(l);
    return 0;
}
```



Duplicação

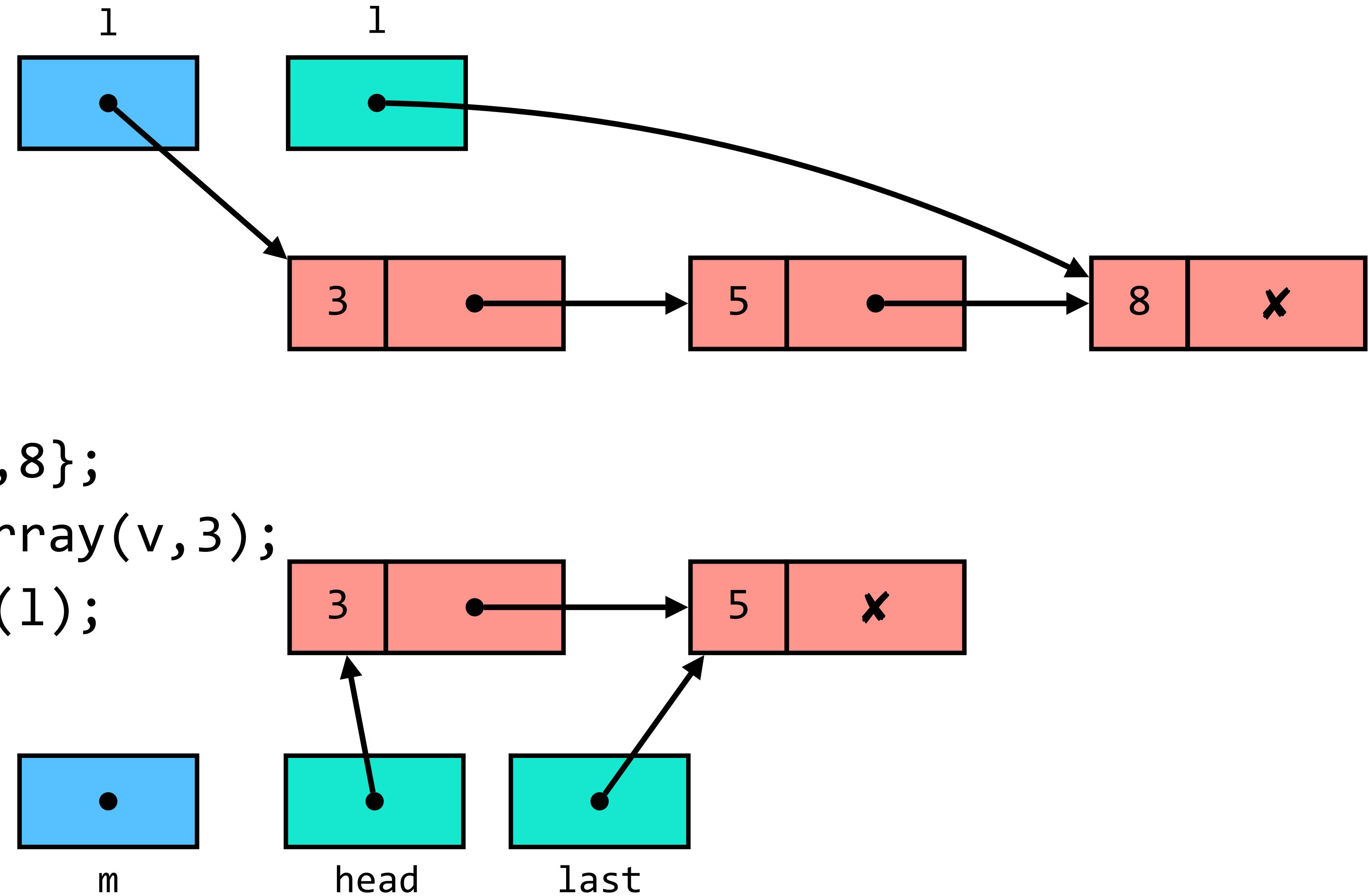


```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    lint m = clone(l);
    return 0;
}
```



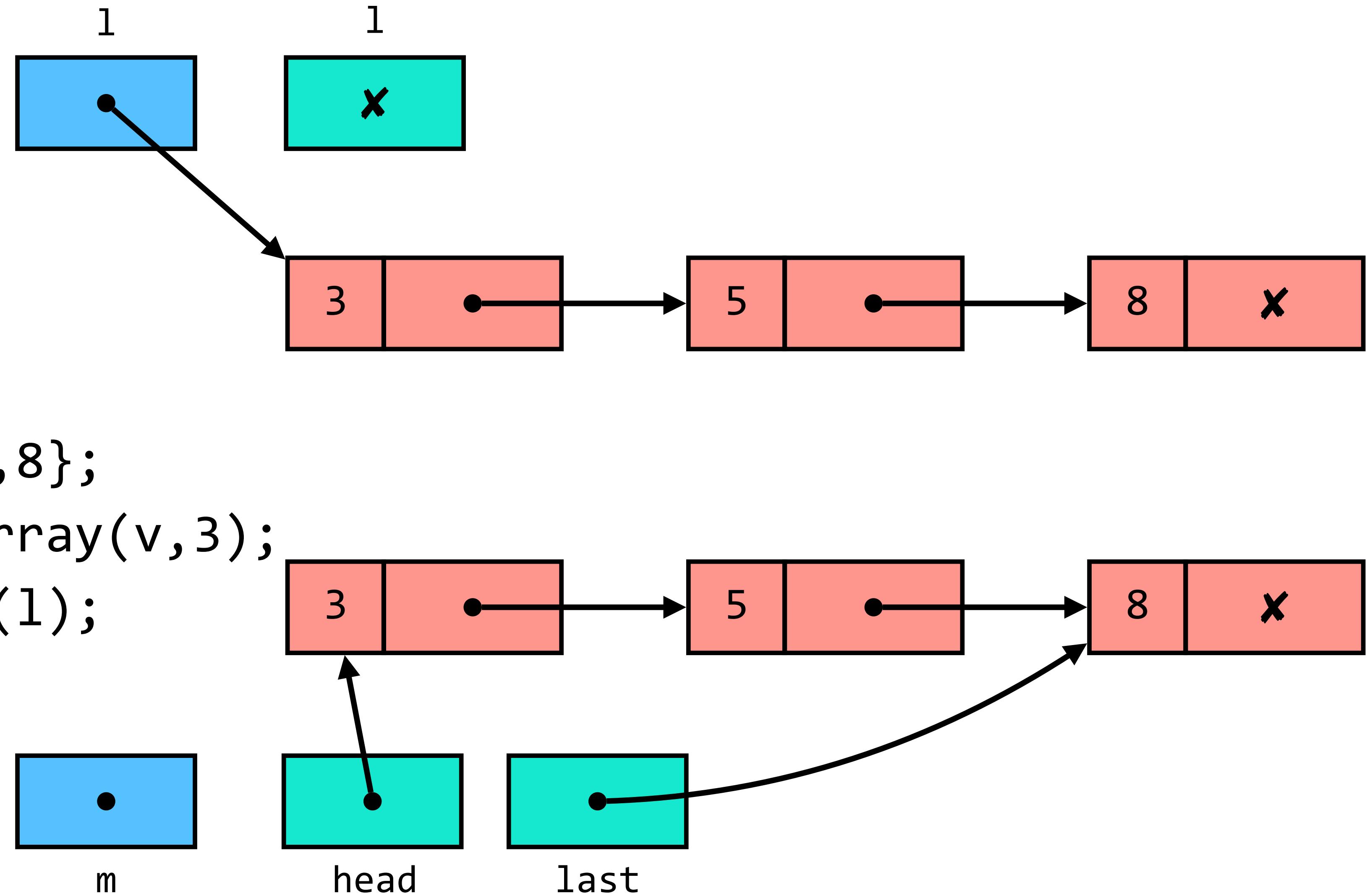
Duplicação

```
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v,3);  
    lint m = clone(l);  
    return 0;  
}
```

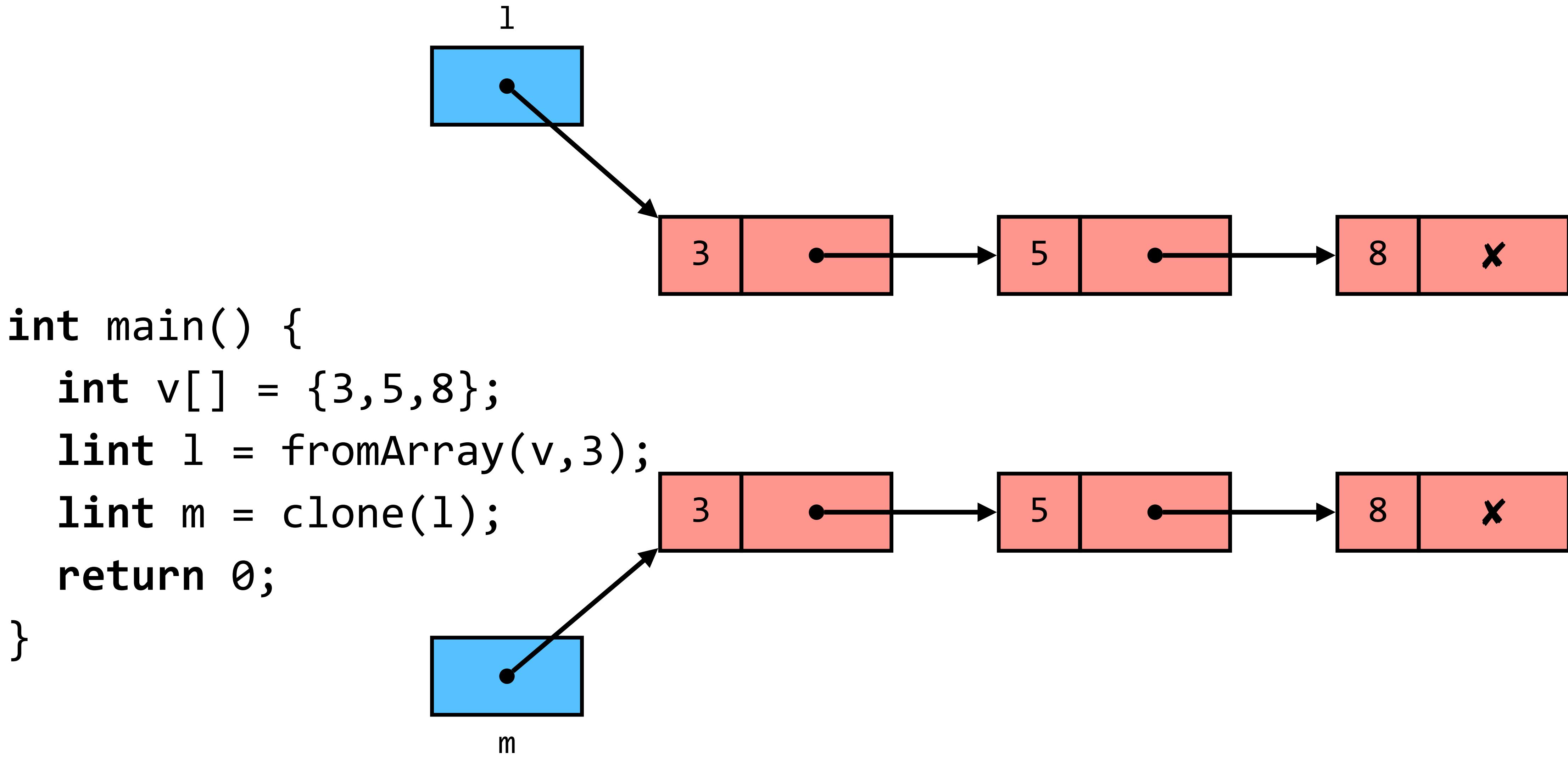


Duplicação

```
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v,3);  
    lint m = clone(l);  
    return 0;  
}
```



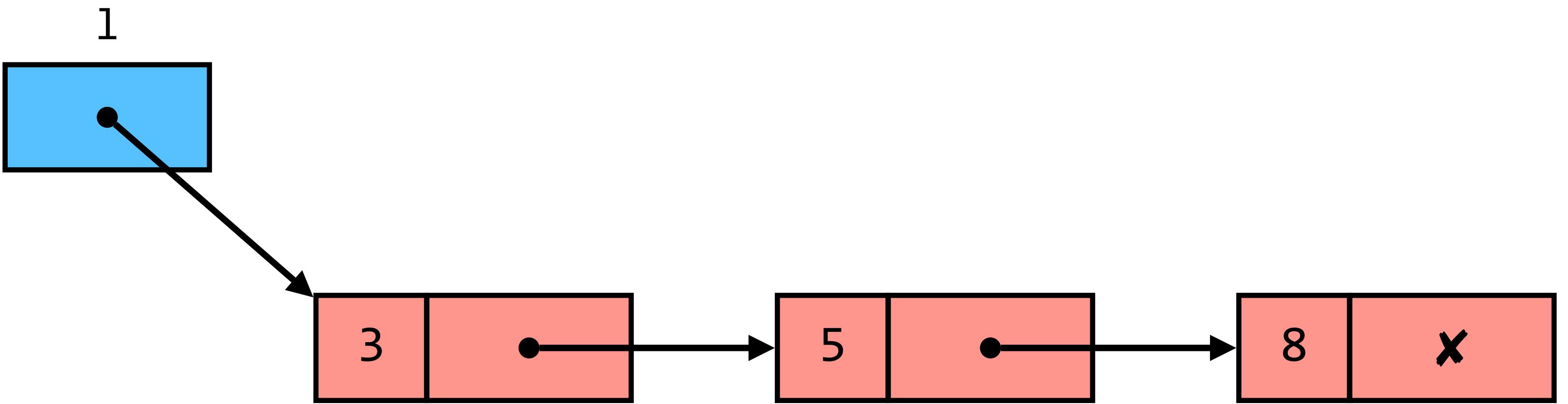
Duplicação



Duplicação

```
lint clone(lint l) {
    lint head = NULL, last = NULL;
    while (l != NULL) {
        if (head == NULL) {
            head = cons(l->valor, NULL);
            last = head;
        } else {
            last->prox = cons(l->valor, NULL);
            last = last->prox;
        }
        l = l->prox;
    }
    return head;
}
```

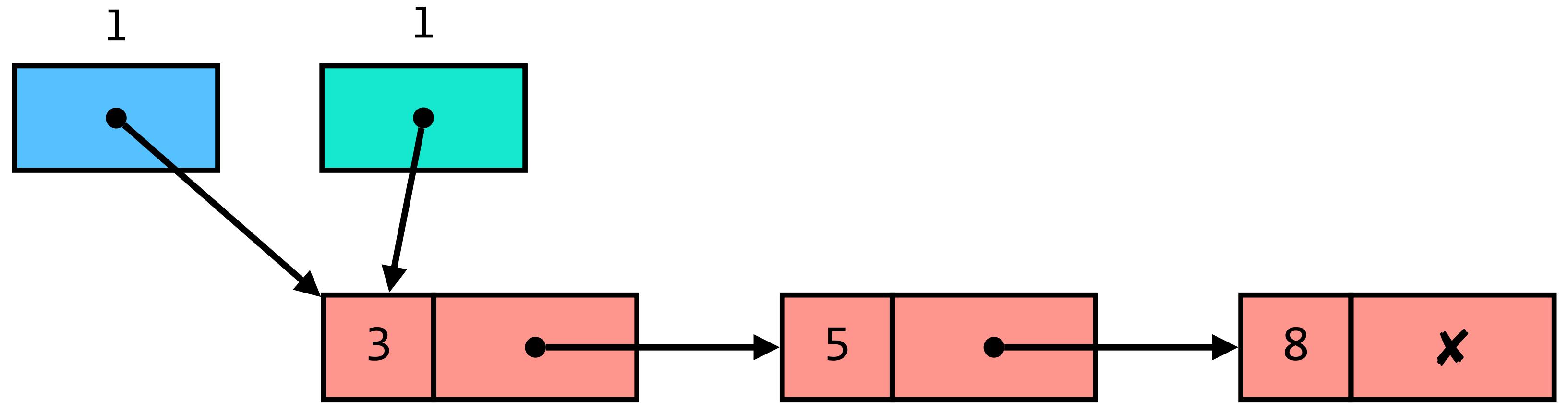
Duplicação



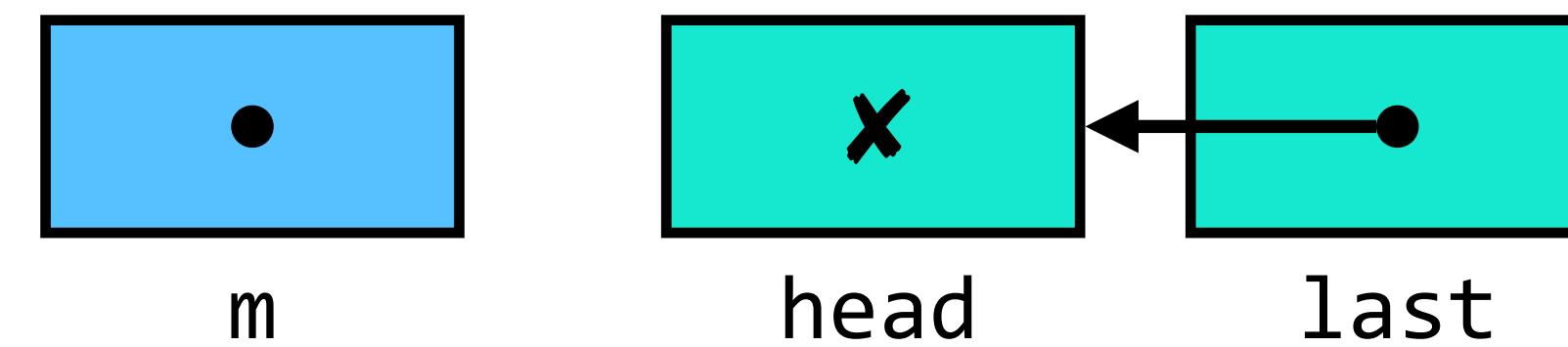
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    lint m = clone(l);
    return 0;
}
```

m

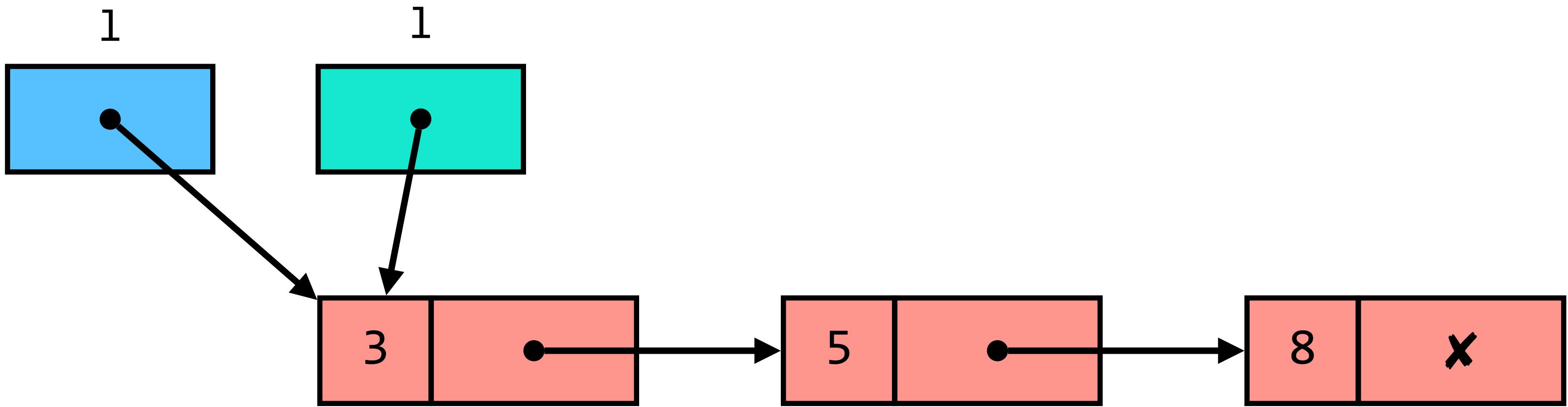
Duplicação



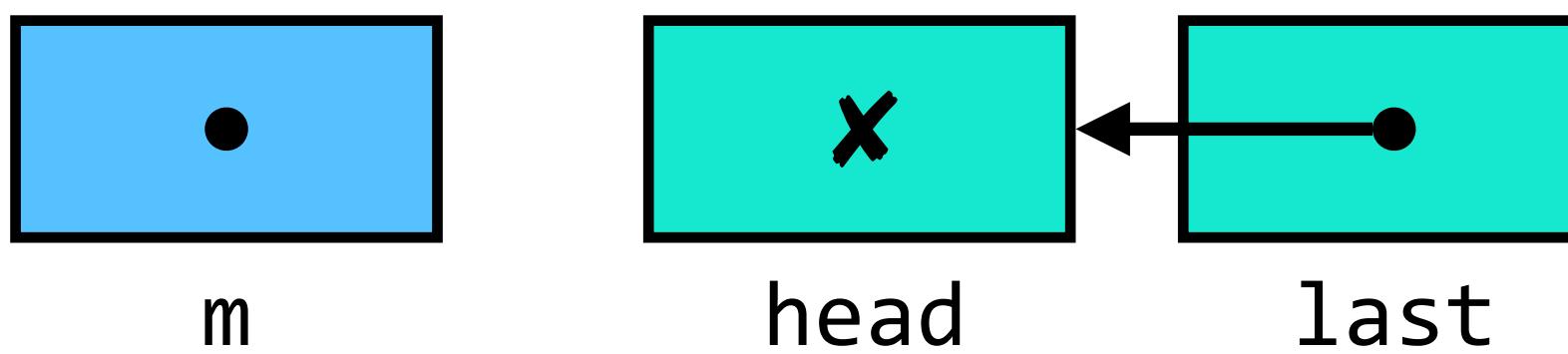
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    lint m = clone(l);
    return 0;
}
```



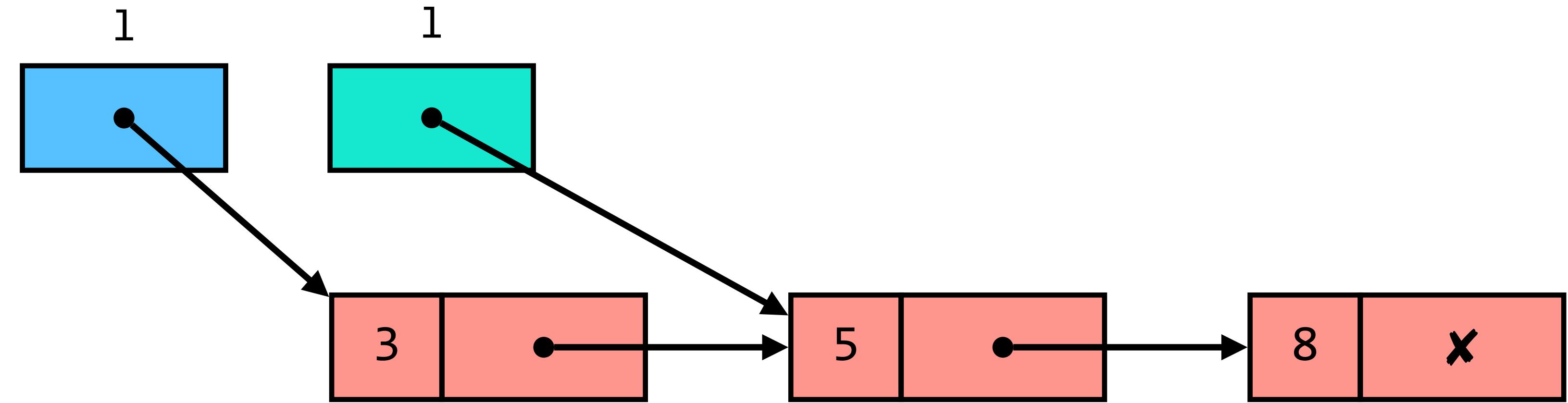
#24 Qual o tipo de last?



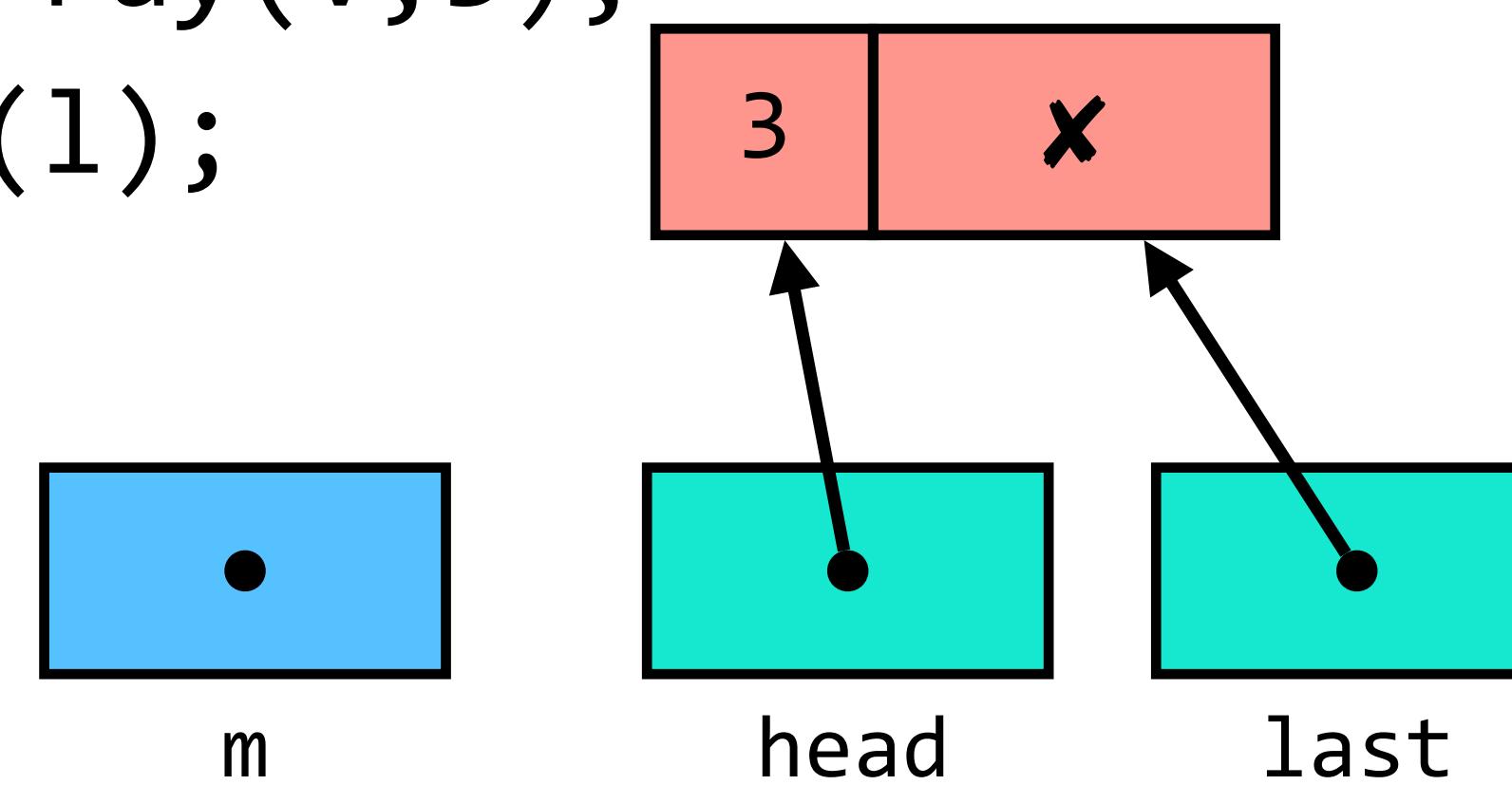
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    lint m = clone(l);
    return 0;
}
```



Duplicação

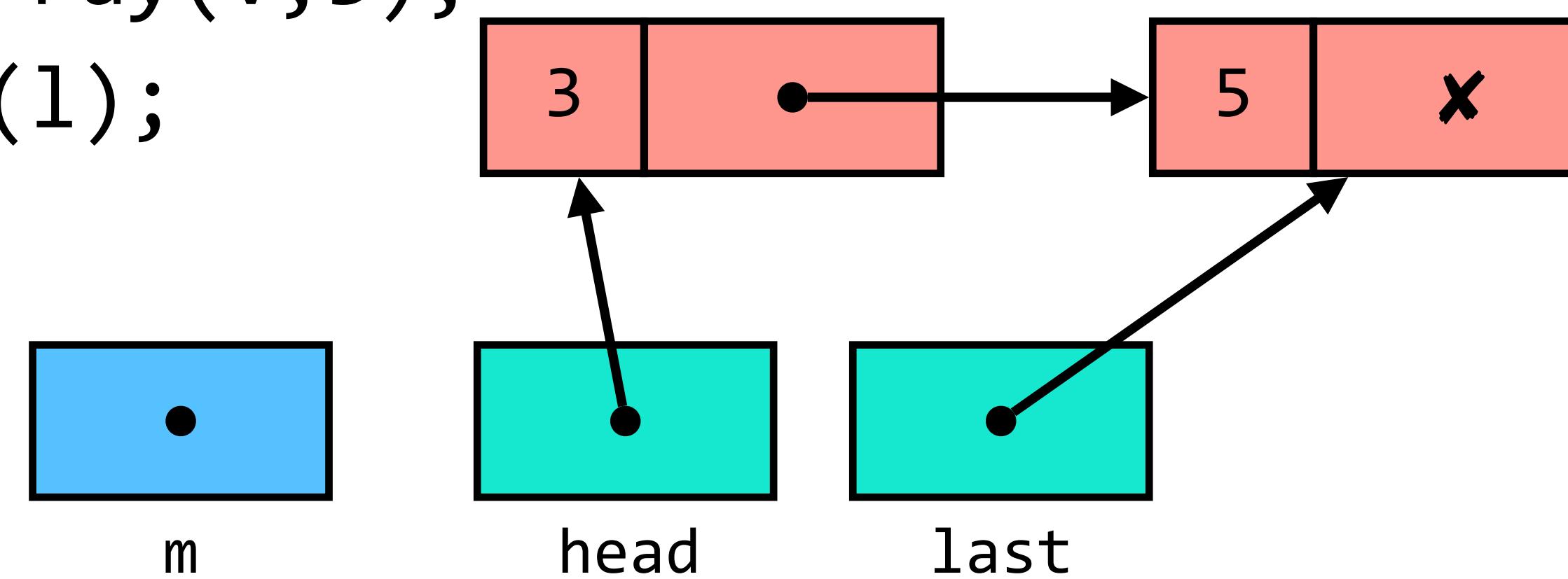
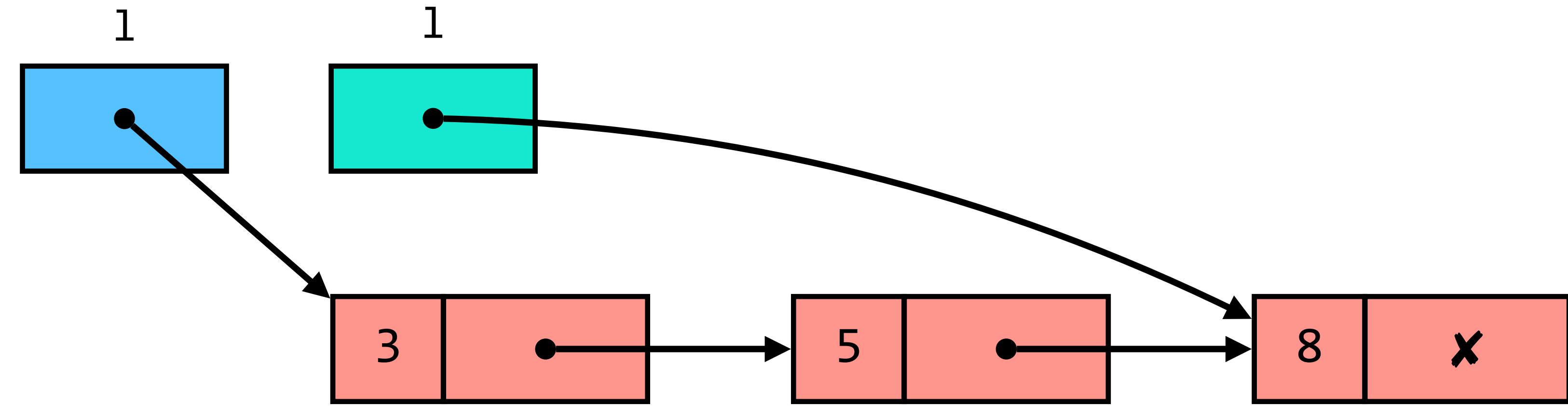


```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    lint m = clone(l);
    return 0;
}
```



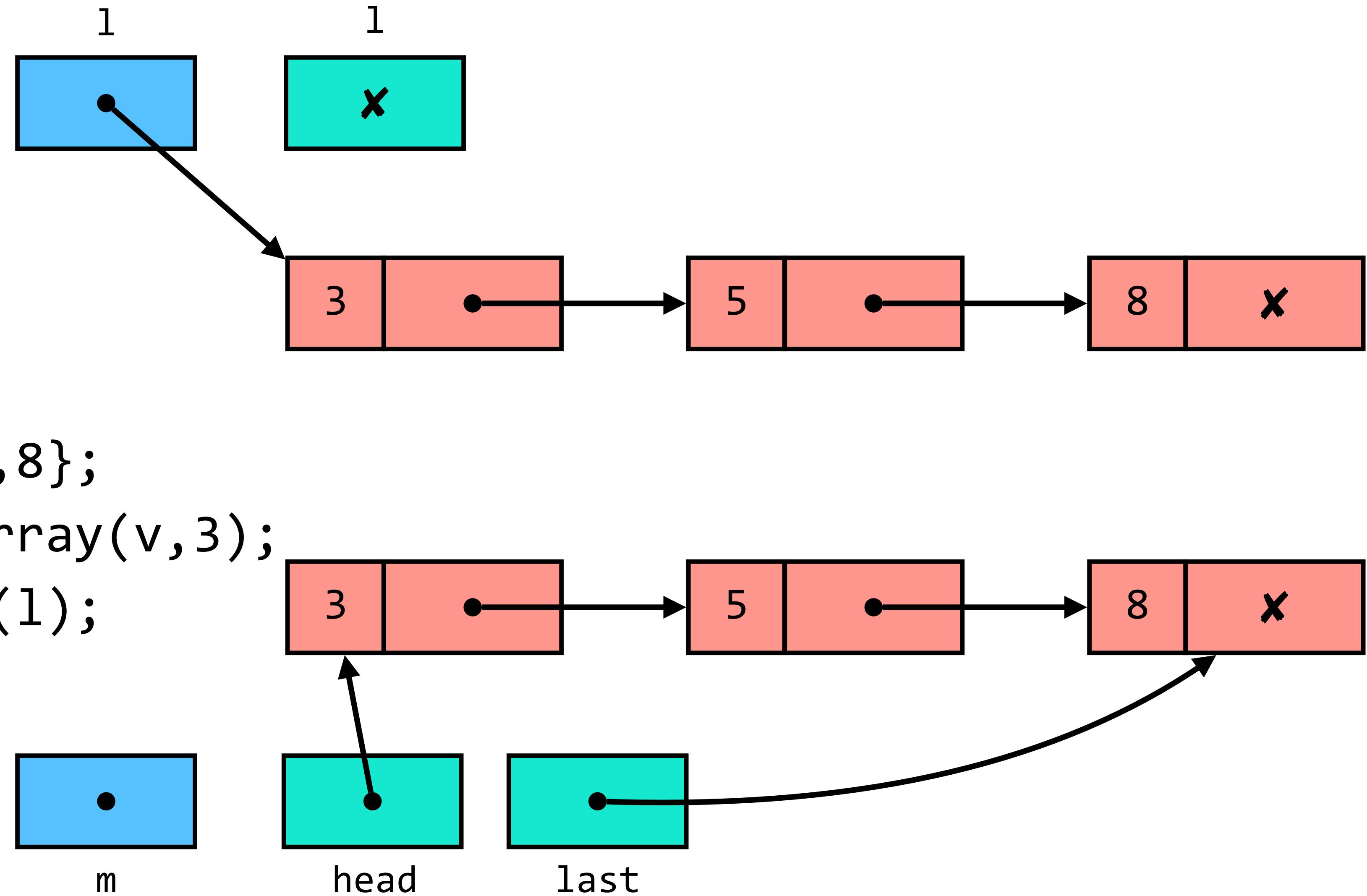
Duplicação

```
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v,3);  
    lint m = clone(l);  
    return 0;  
}
```

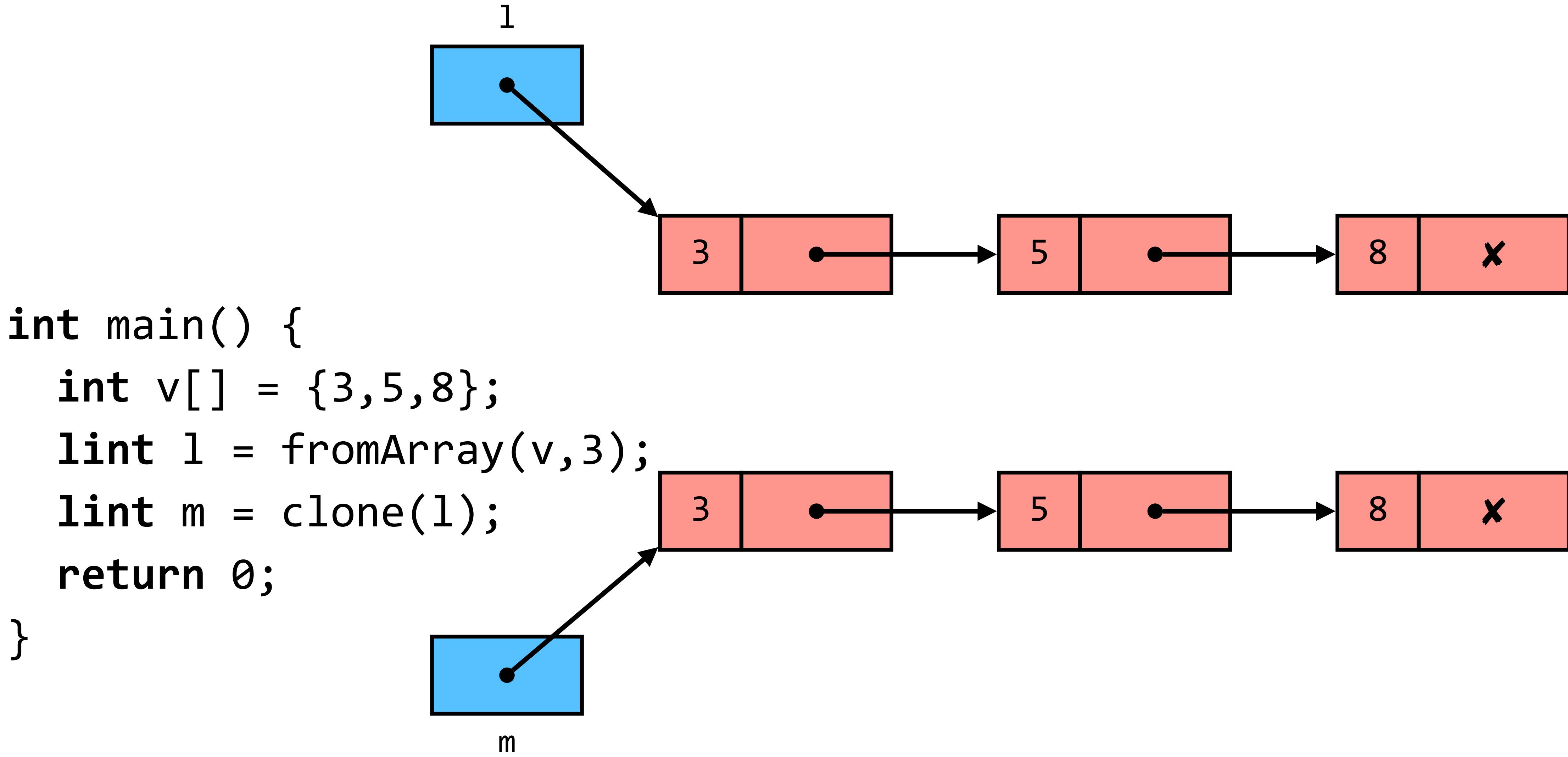


Duplicação

```
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v,3);  
    lint m = clone(l);  
    return 0;  
}
```



Duplicação

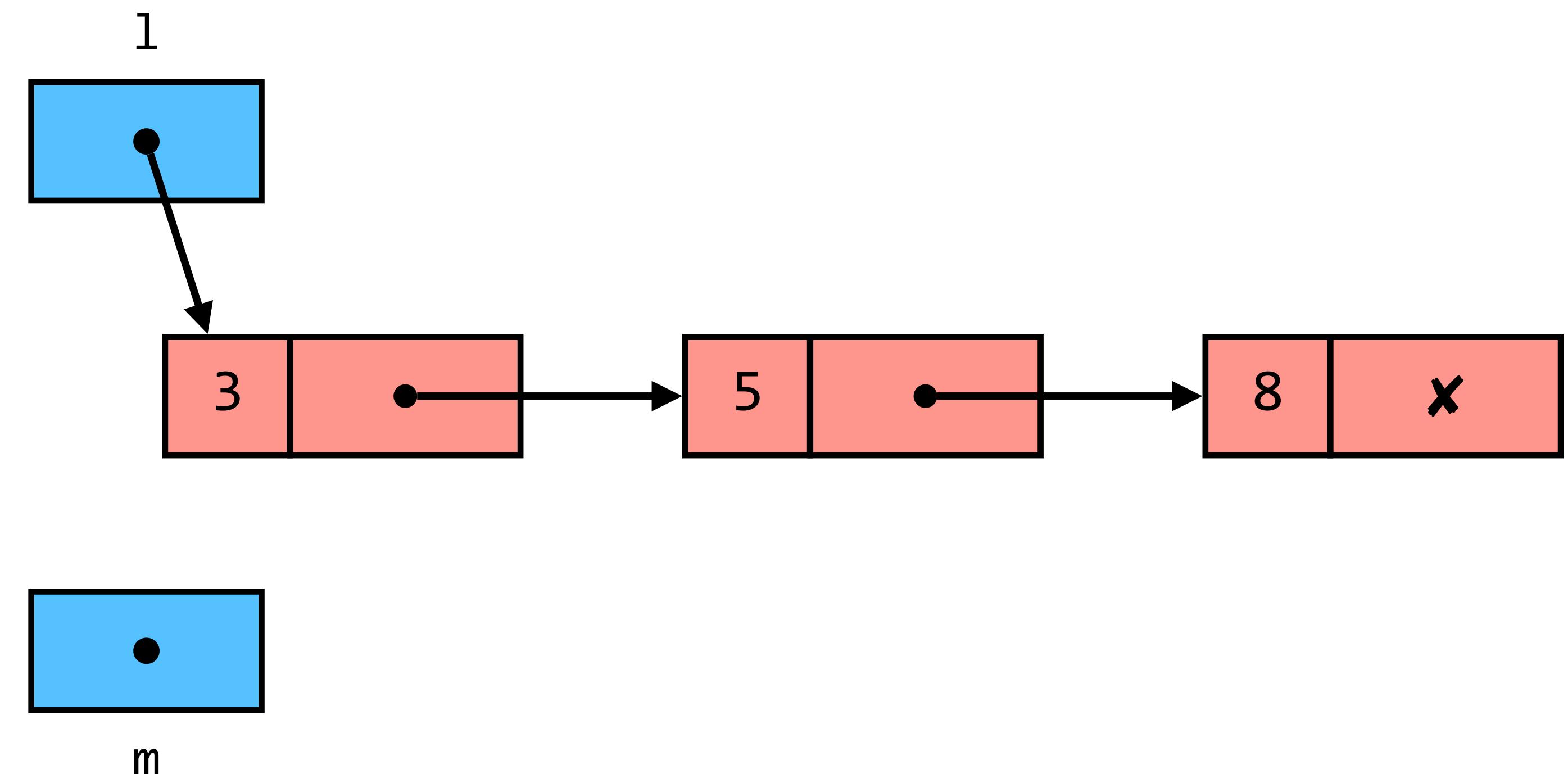


Duplicação

```
lint clone(lint l) {
    lint head = NULL, *last = &head;
    while (l != NULL) {
        *last = cons(l->valor, NULL);
        last = &((*last)->prox);
        l = l->prox;
    }
    return head;
}
```

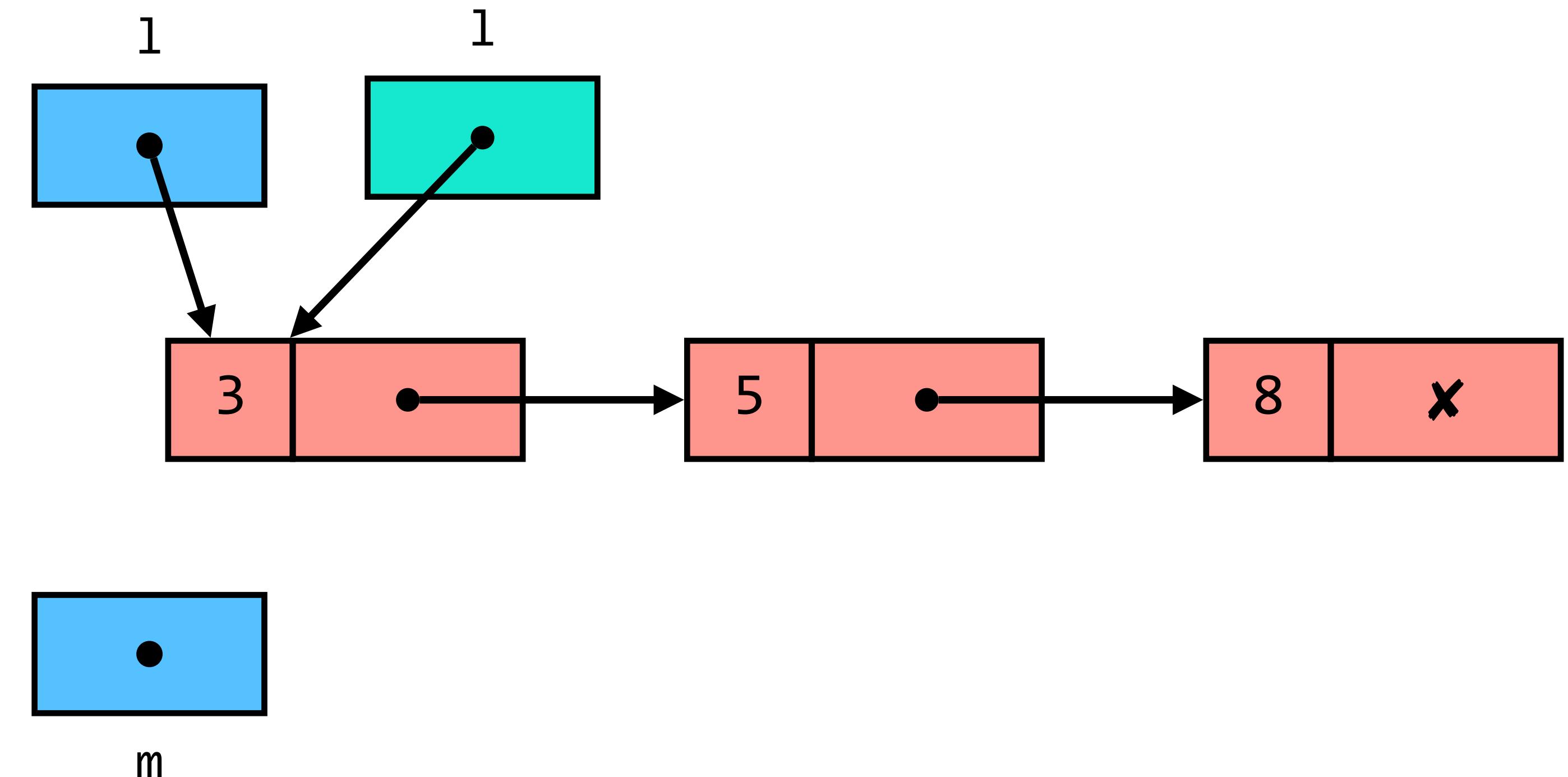
Pesquisa ordenada

```
lint search(int x, lint l) {  
    while (l != NULL && l->valor < x) l = l->prox;  
    if (l != NULL && l->valor == x) return l;  
    else return NULL;  
}  
  
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v, 3);  
    lint m = search(5, l);  
    return 0;  
}
```



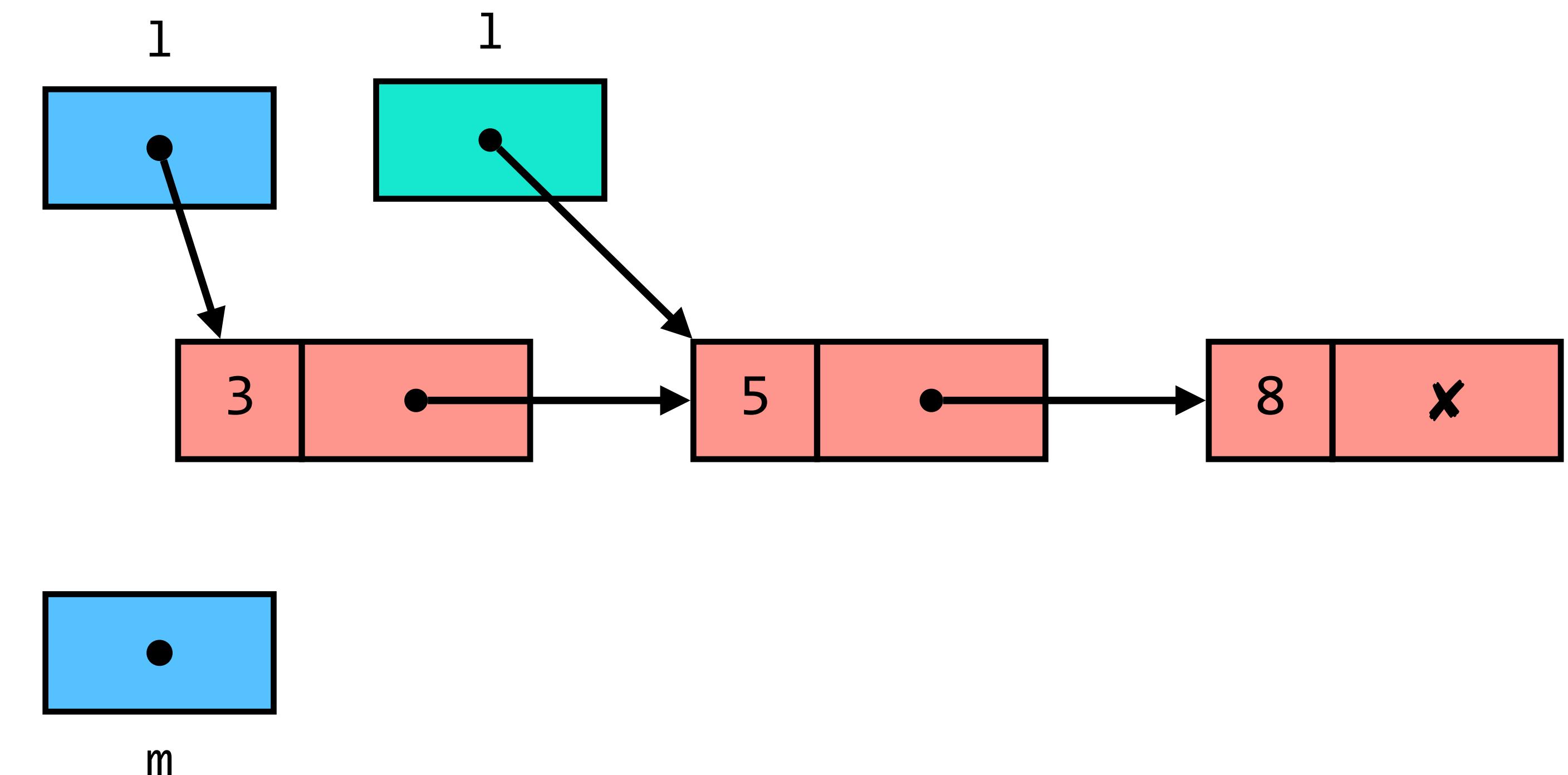
Pesquisa ordenada

```
lint search(int x, lint l) {  
    while (l != NULL && l->valor < x) l = l->prox;  
    if (l != NULL && l->valor == x) return l;  
    else return NULL;  
}  
  
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v, 3);  
    lint m = search(5, l);  
    return 0;  
}
```



Pesquisa ordenada

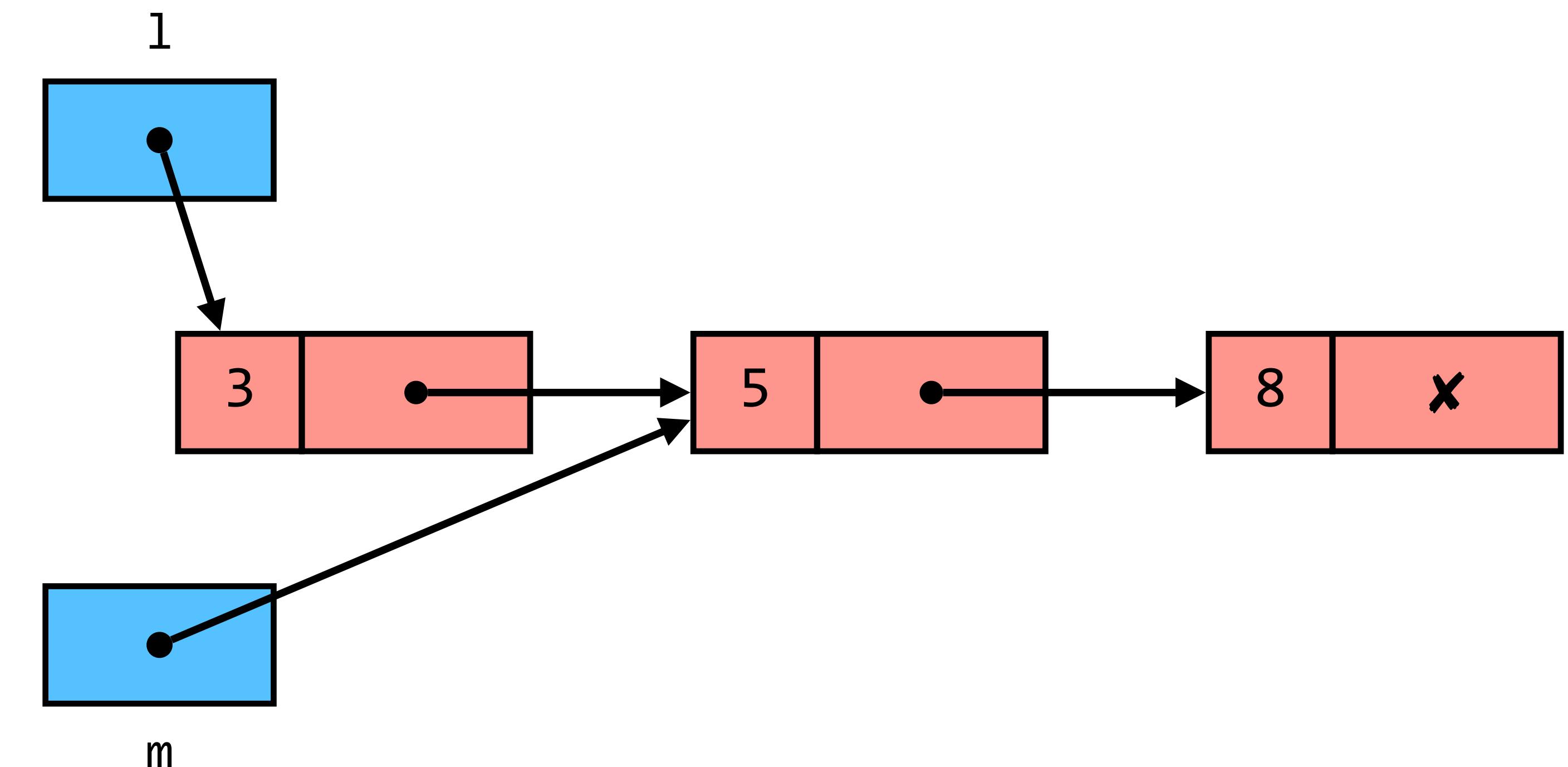
```
lint search(int x, lint l) {  
    while (l != NULL && l->valor < x) l = l->prox;  
    if (l != NULL && l->valor == x) return l;  
    else return NULL;  
}  
  
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v, 3);  
    lint m = search(5, l);  
    return 0;  
}
```



Pesquisa ordenada

```
lint search(int x, lint l) {  
    while (l != NULL && l->valor < x) l = l->prox;  
    if (l != NULL && l->valor == x) return l;  
    else return NULL;  
}
```

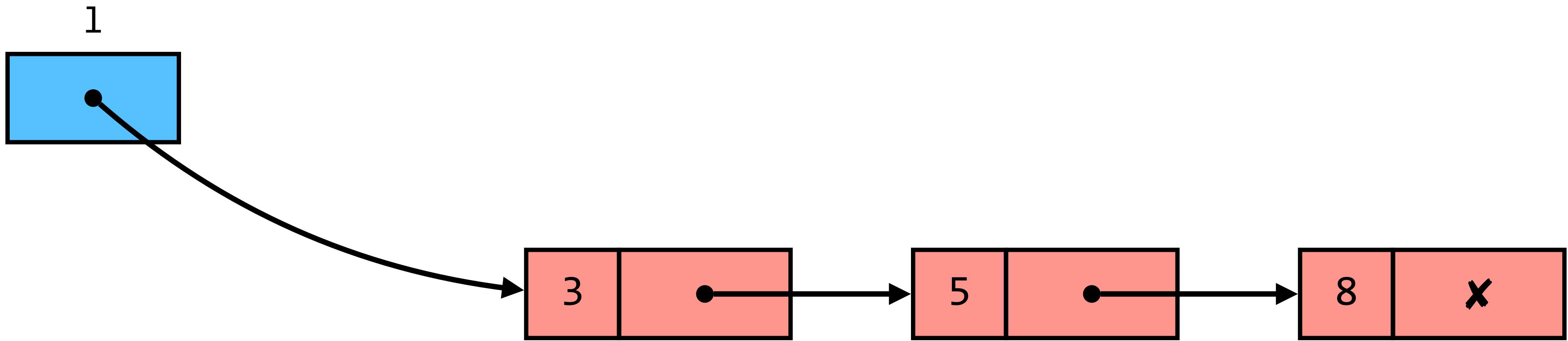
```
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v, 3);  
    lint m = search(5, l);  
    return 0;  
}
```



Inserção ordenada

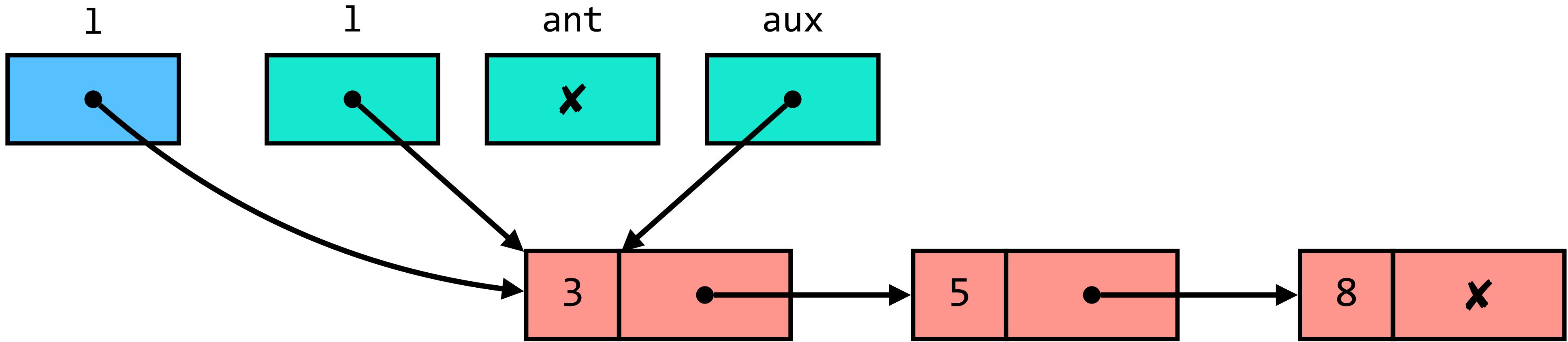
```
lint insert(int x, lint l) {  
    ...  
}
```

Inserção ordenada



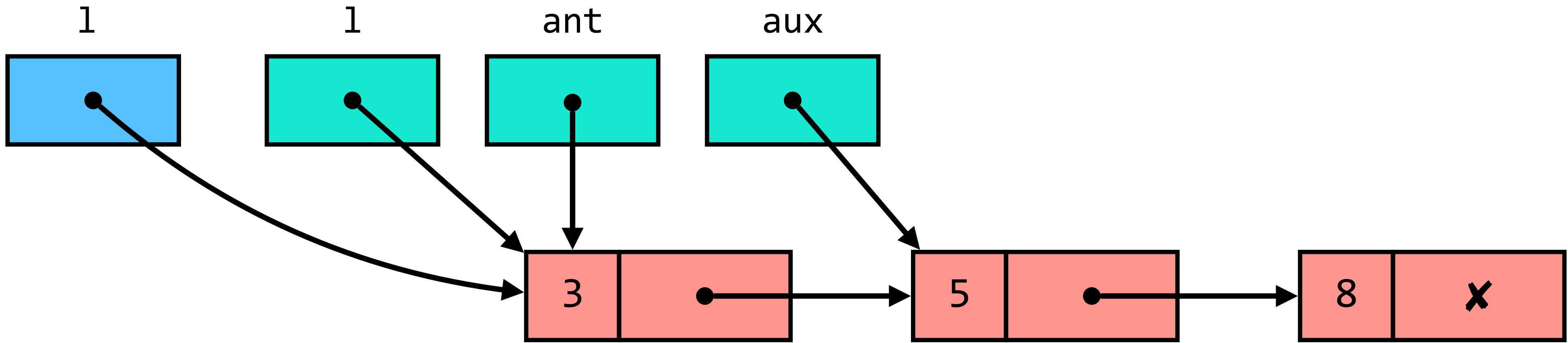
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = insert(6,l);
    return 0;
}
```

Inserção ordenada



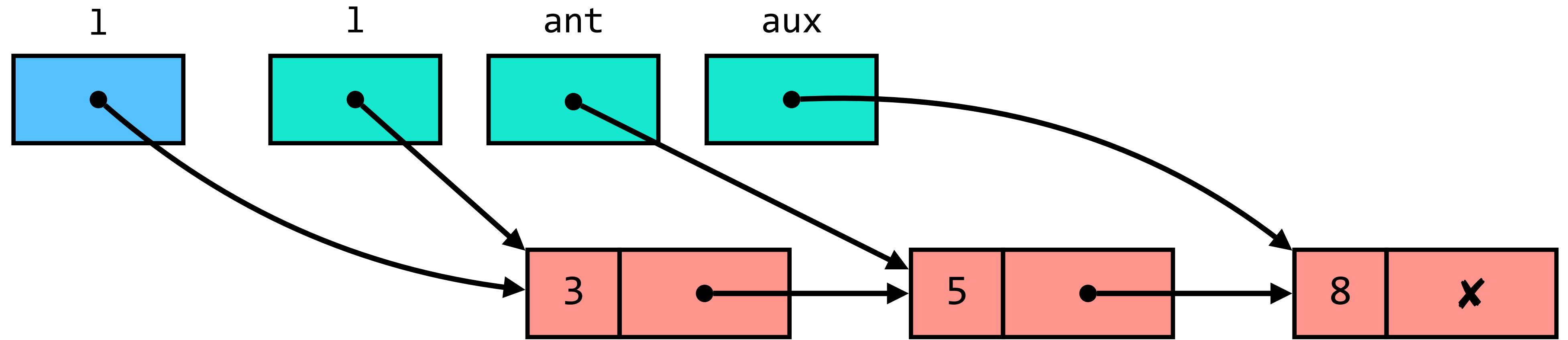
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = insert(6,l);
    return 0;
}
```

Inserção ordenada



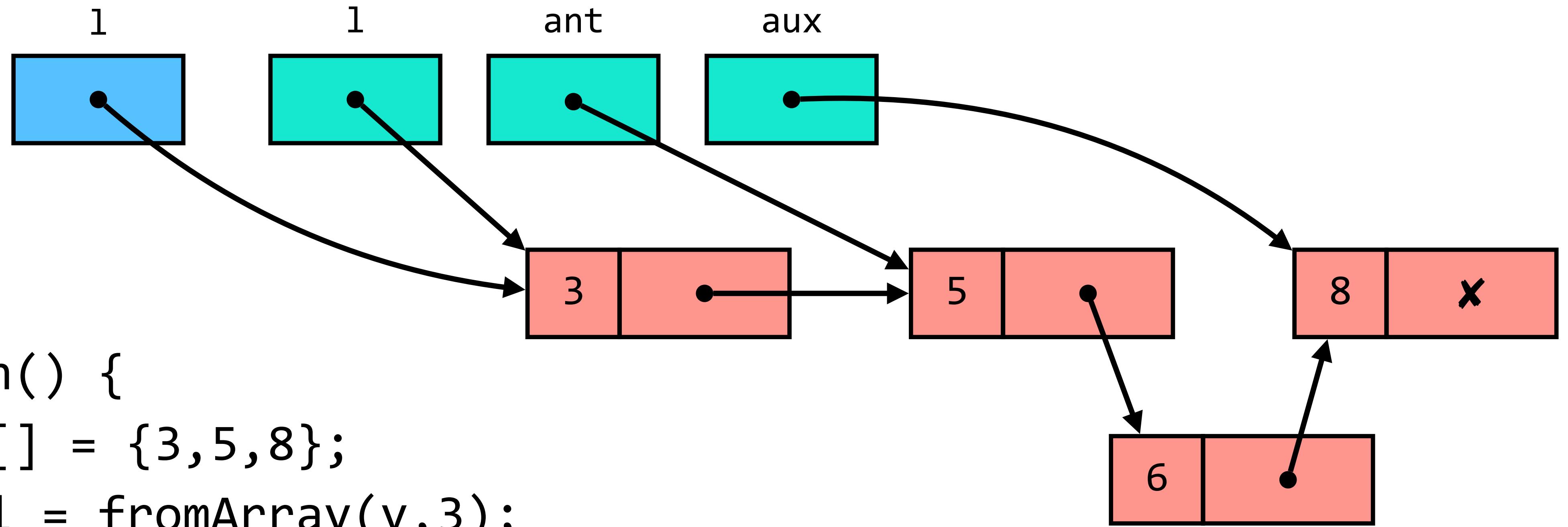
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = insert(6,l);
    return 0;
}
```

Inserção ordenada



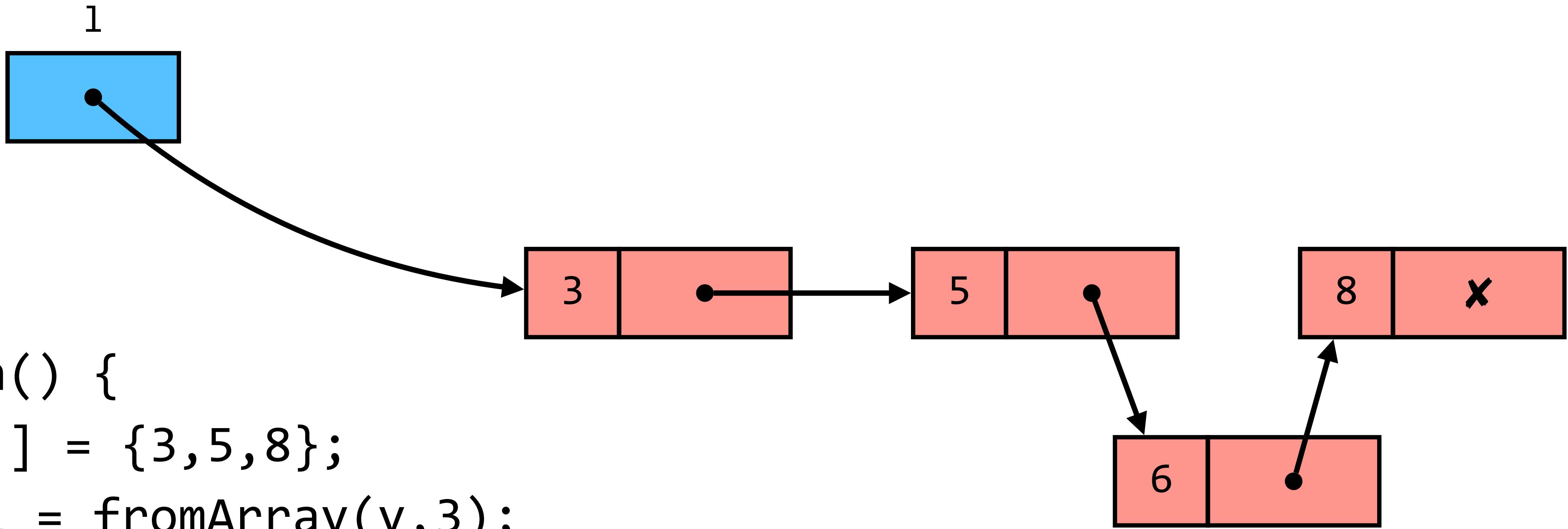
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = insert(6,l);
    return 0;
}
```

Inserção ordenada



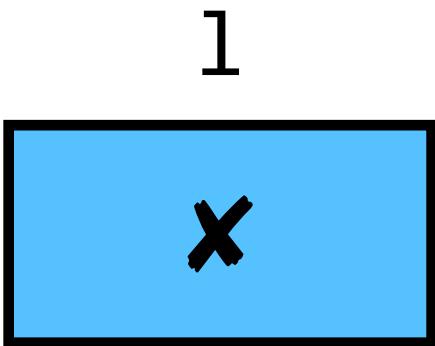
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = insert(6,l);
    return 0;
}
```

Inserção ordenada



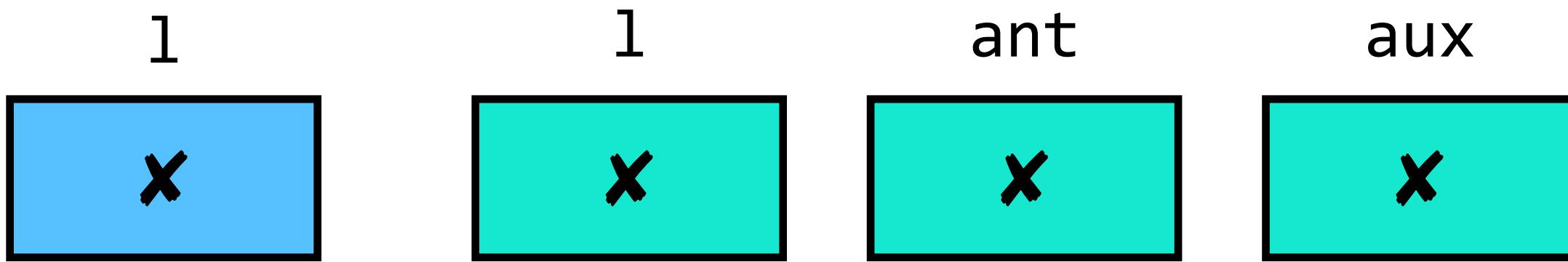
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = insert(6,l);
    return 0;
}
```

Inserção ordenada



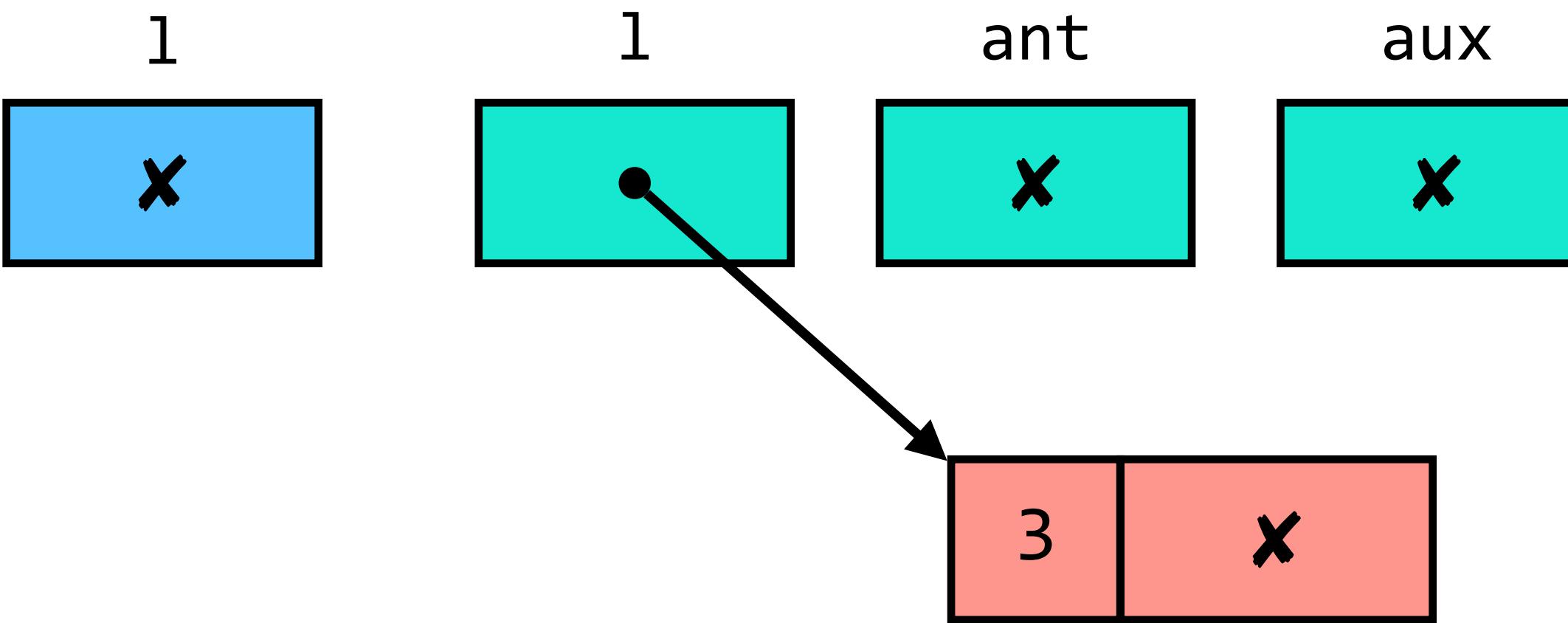
```
int main() {
    lnt l = NULL;
    l = insert(3,l);
    return 0;
}
```

Inserção ordenada



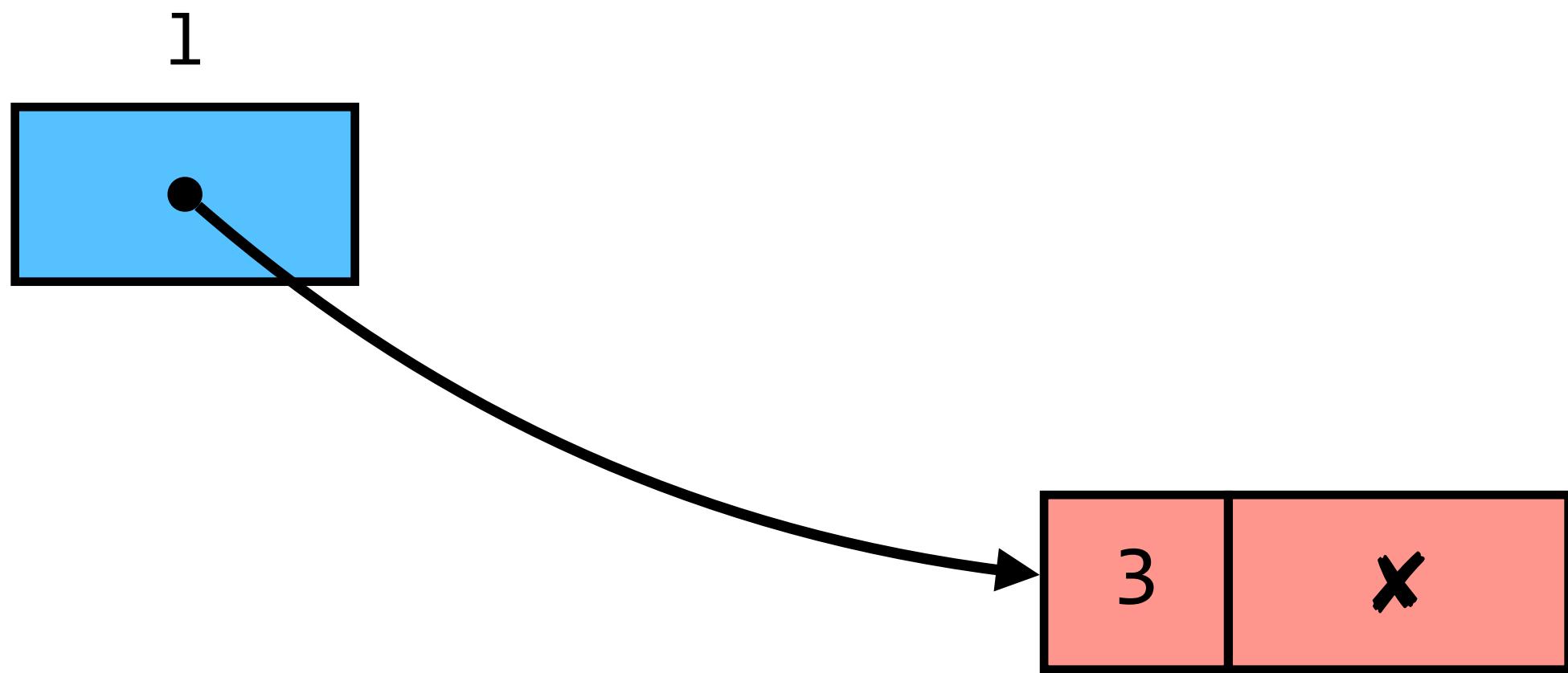
```
int main() {
    lint l = NULL;
    l = insert(3,l);
    return 0;
}
```

Inserção ordenada



```
int main() {
    lint l = NULL;
    l = insert(3,l);
    return 0;
}
```

Inserção ordenada

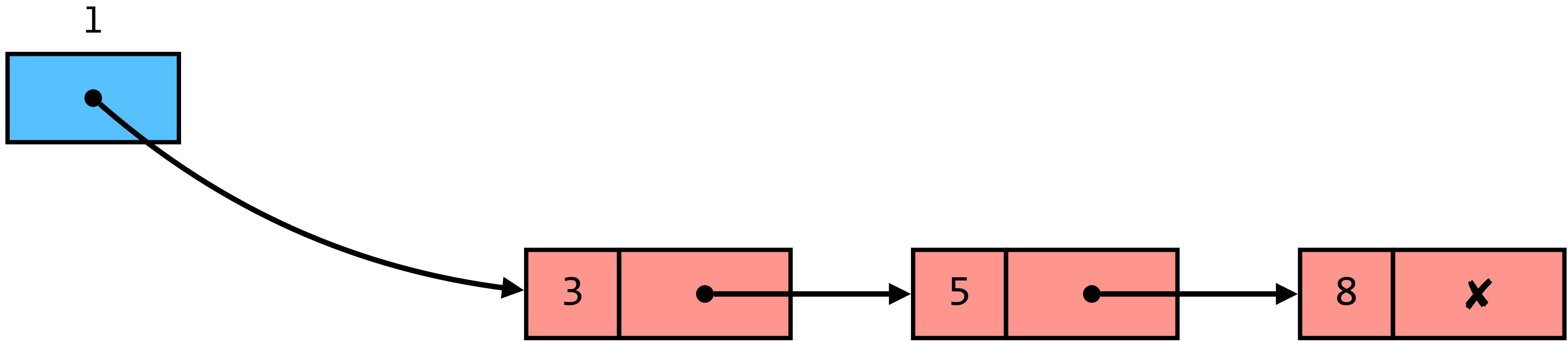


```
int main() {
    lint l = NULL;
    l = insert(3,l);
    return 0;
}
```

Inserção ordenada

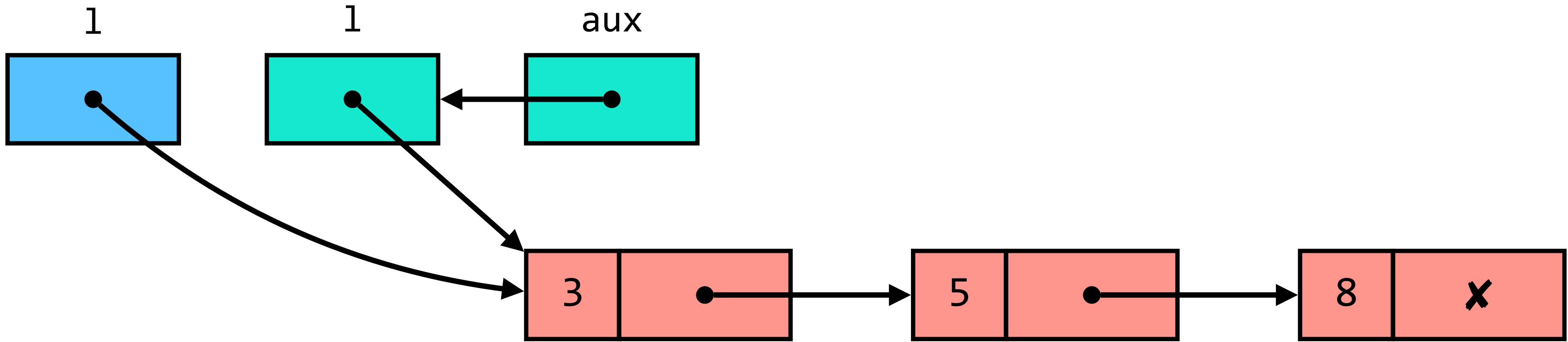
```
lint insert(int x, lint l) {
    lint ant = NULL, aux = l;
    while (aux != NULL && aux->valor < x) {
        ant = aux;
        aux = aux->prox;
    }
    if (ant == NULL) l = cons(x, l);
    else ant->prox = cons(x, aux);
    return l;
}
```

Inserção ordenada



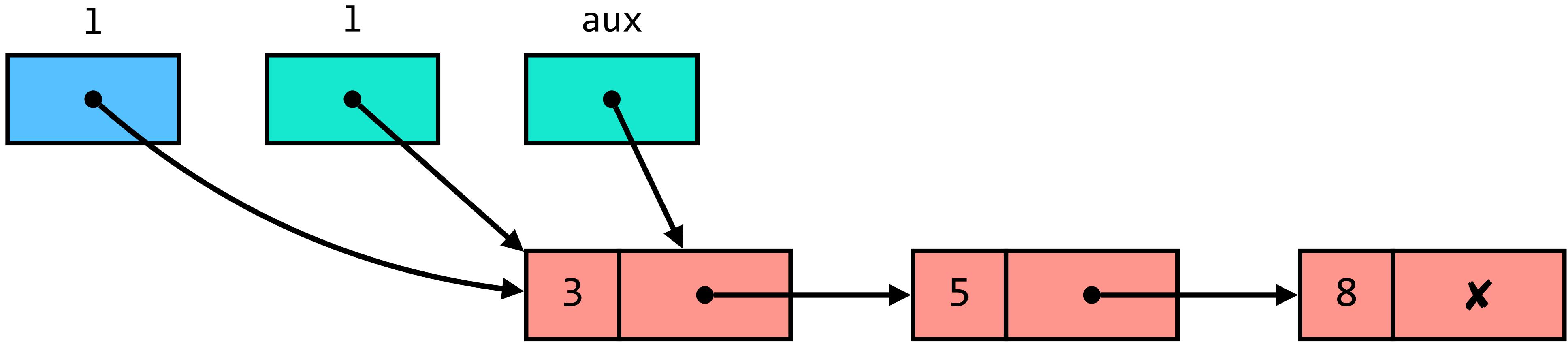
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = insert(6,l);
    return 0;
}
```

Inserção ordenada



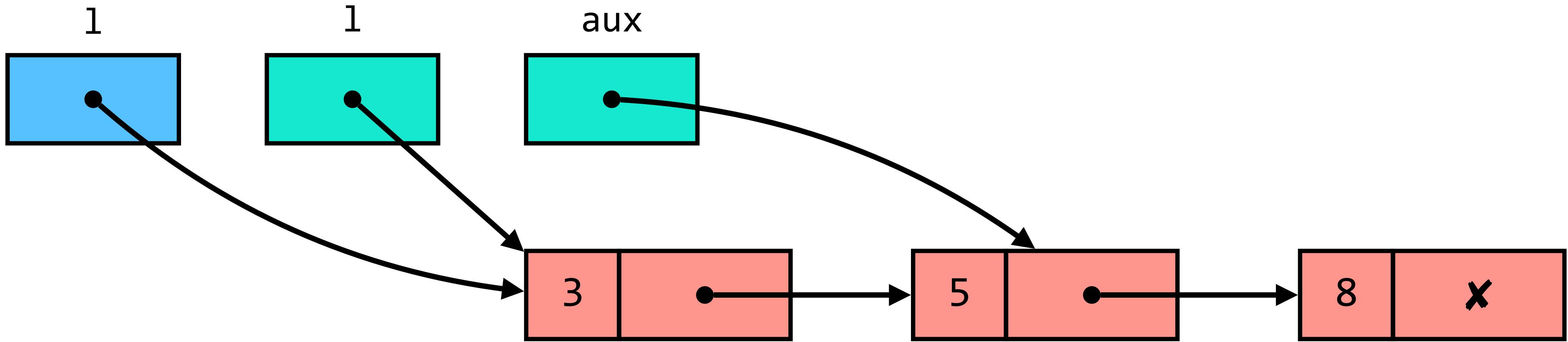
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = insert(6,l);
    return 0;
}
```

Inserção ordenada



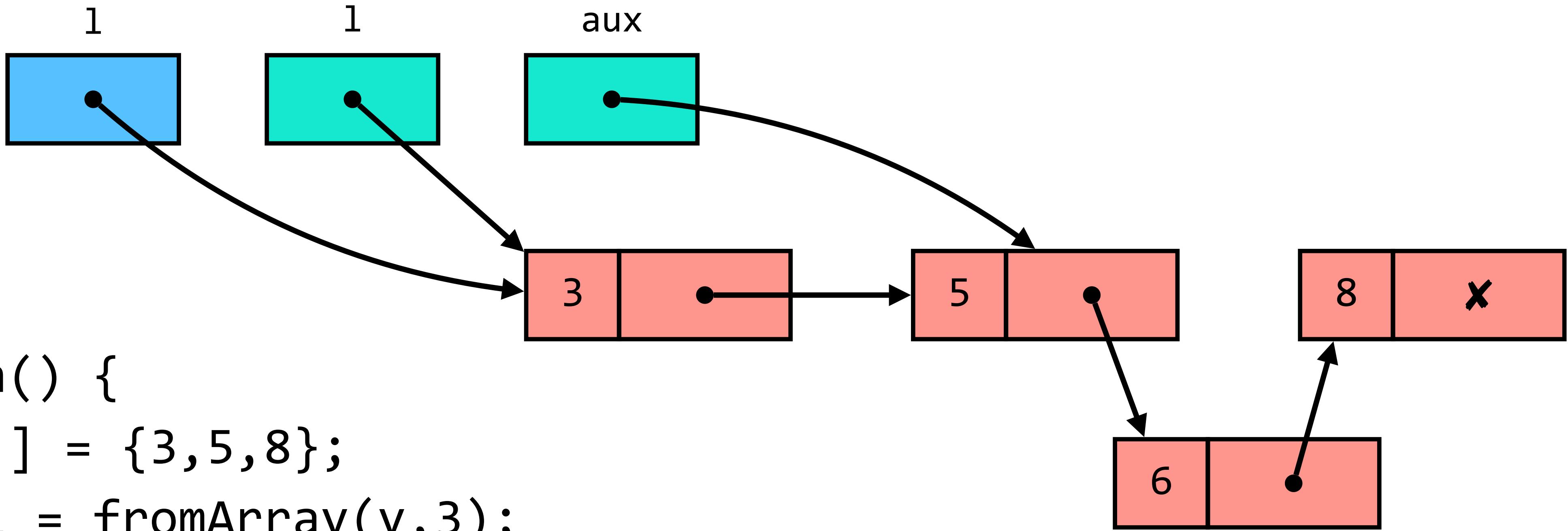
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = insert(6,l);
    return 0;
}
```

Inserção ordenada



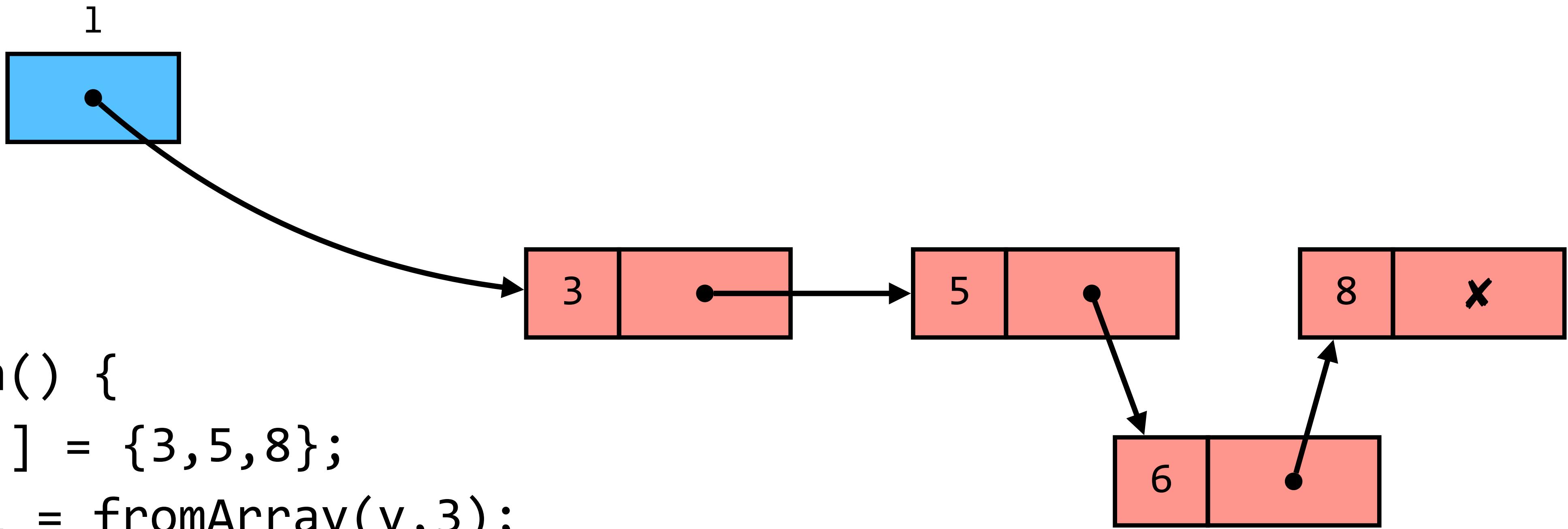
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = insert(6,l);
    return 0;
}
```

Inserção ordenada



```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = insert(6,l);
    return 0;
}
```

Inserção ordenada



```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = insert(6,l);
    return 0;
}
```

Inserção ordenada

```
lint insert(int x, lint l) {
    lint *aux = &l;
    while ((*aux) != NULL && (*aux)->valor < x) {
        aux = &((*aux)->prox);
    }
    *aux = cons(x,*aux);
    return l;
}
```

#25 Qual a lista l depois do insert?

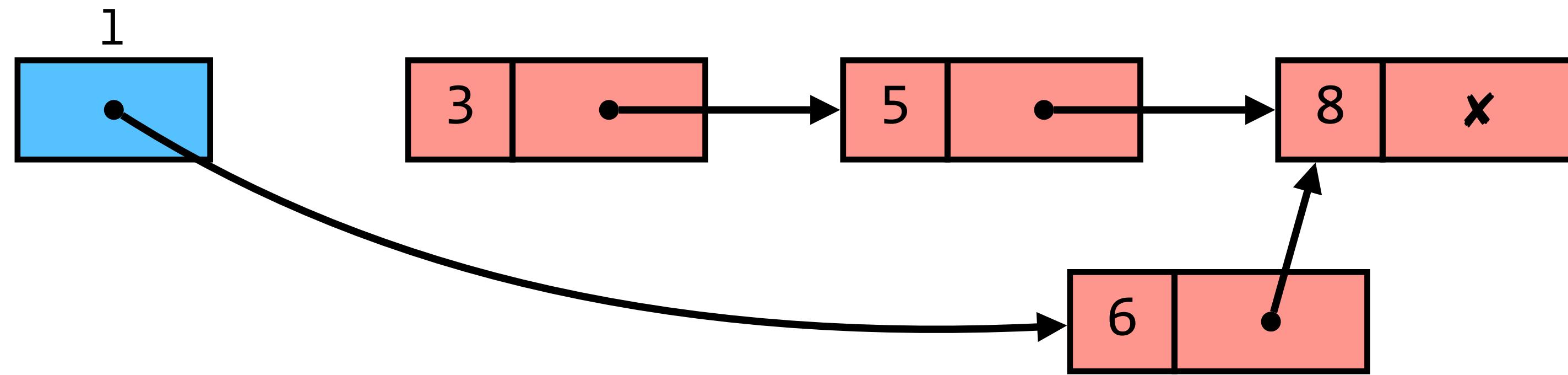
```
lint insert(int x, lint l) {
    lint *aux = &l;
    while ((*aux) != NULL && (*aux)->valor < x) {
        *aux = (*aux)->prox; // aux = &((*aux)->prox);
    }
    *aux = cons(x,*aux);
    return l;
}
```

```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = insert(6,l);
    return 0;
}
```



#25 Qual a lista l depois do insert?

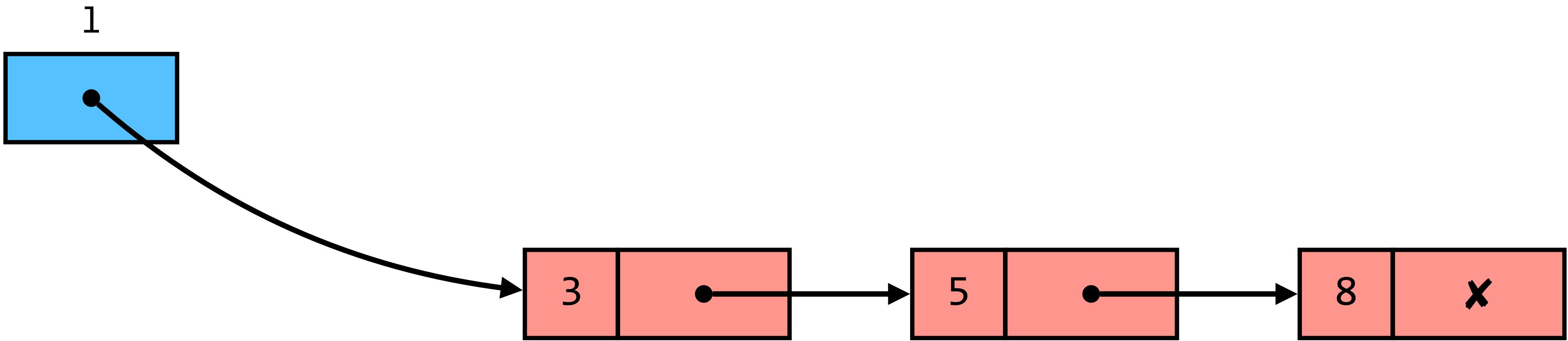
```
lint insert(int x, lint l) {  
    lint *aux = &l;  
    while ((*aux) != NULL && (*aux)->valor < x) {  
        *aux = (*aux)->prox; // aux = &((*aux)->prox);  
    }  
    *aux = cons(x,*aux);  
    return l;  
}  
  
int main() {  
    int v[] = {3,5,8};  
    lint l = fromArray(v,3);  
    l = insert(6,l);  
    return 0;  
}
```



Remoção ordenada

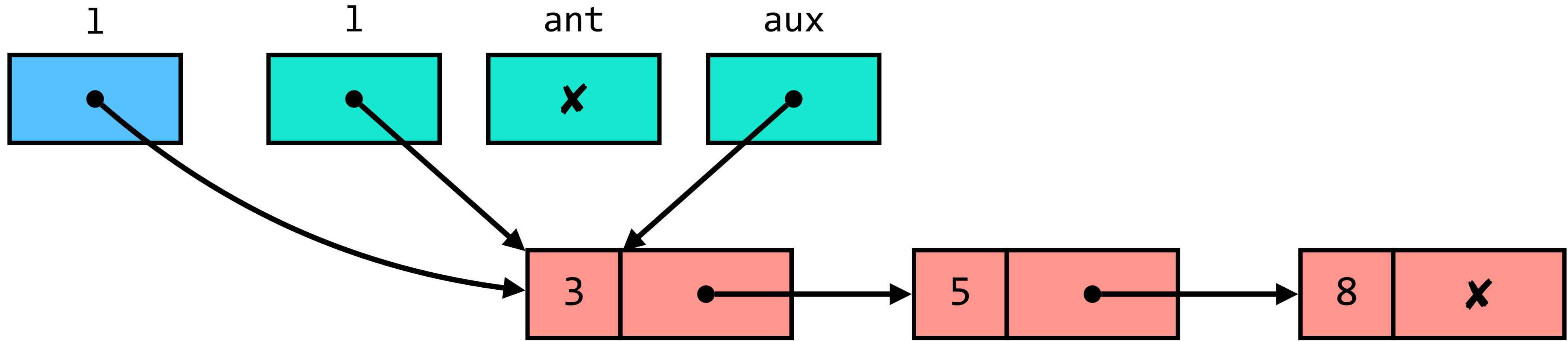
```
lint delete(int x, lint l) {  
    ...  
}
```

Remoção ordenada



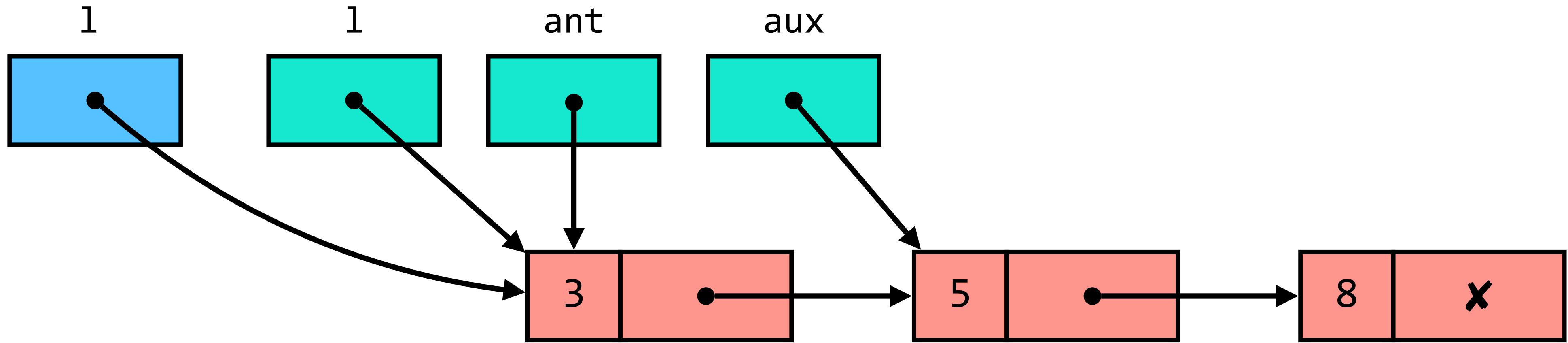
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = delete(5,l);
    return 0;
}
```

Remoção ordenada



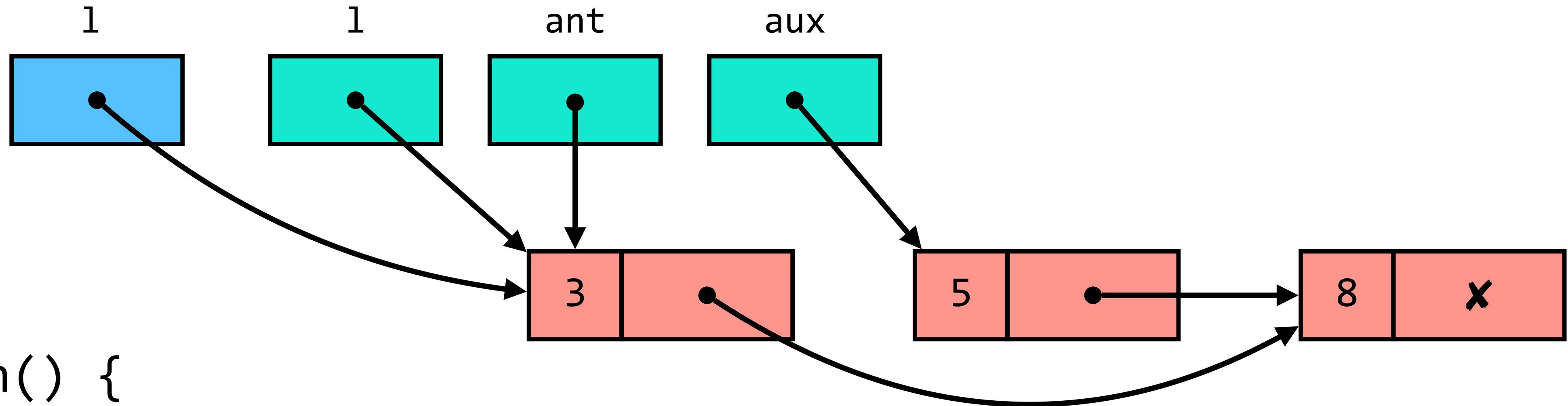
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = delete(5,l);
    return 0;
}
```

Remoção ordenada



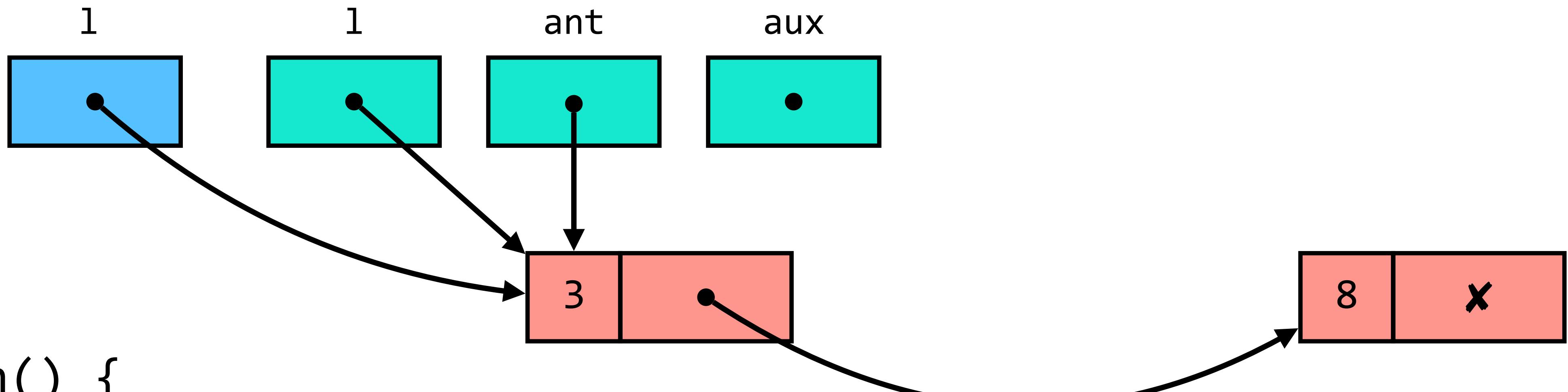
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = delete(5,l);
    return 0;
}
```

Remoção ordenada



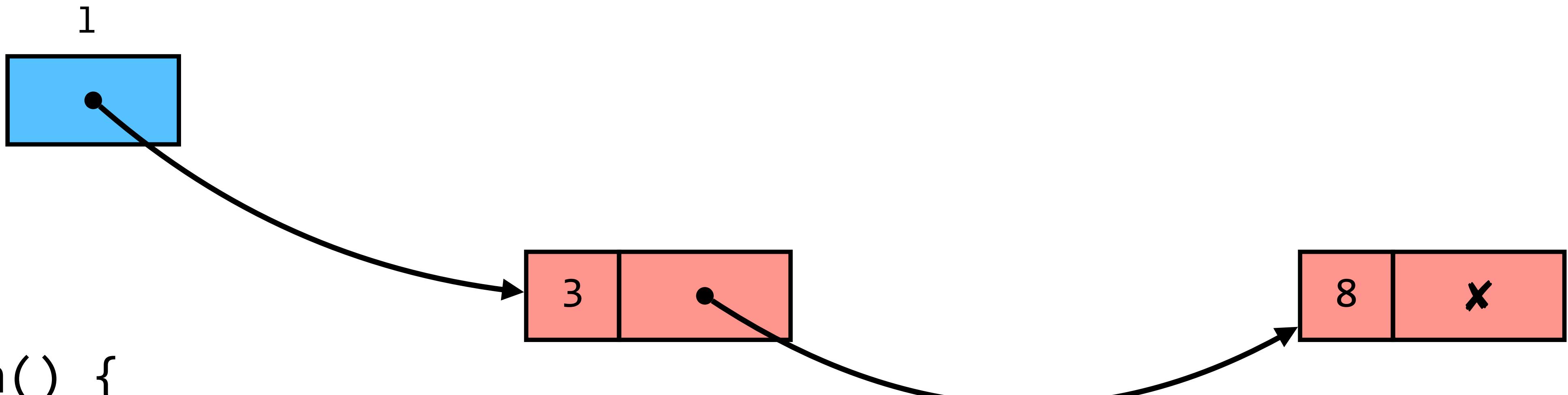
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = delete(5,l);
    return 0;
}
```

Remoção ordenada



```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = delete(5,l);
    return 0;
}
```

Remoção ordenada

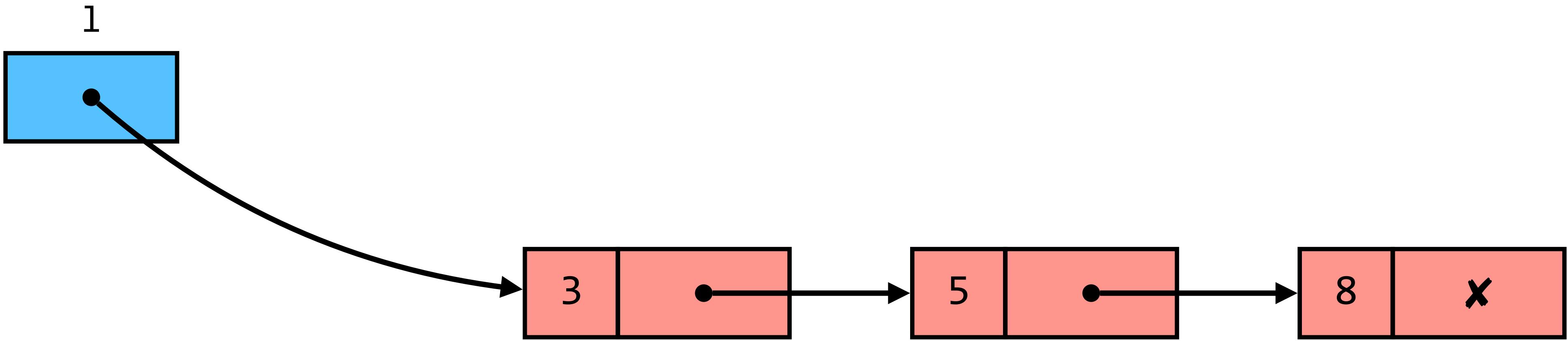


```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = delete(5,l);
    return 0;
}
```

Remoção ordenada

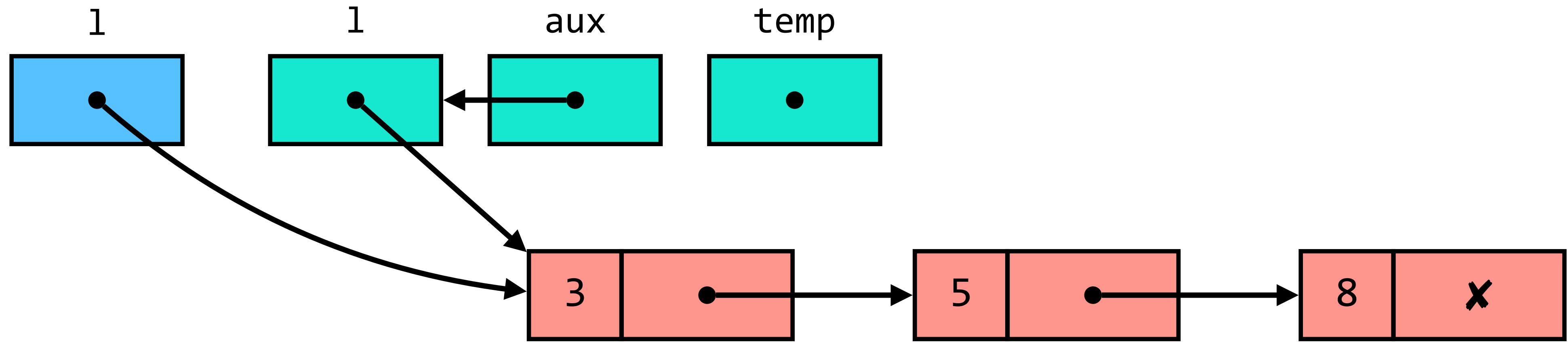
```
lint delete(int x, lint l) {
    lint ant = NULL, aux = l;
    while (aux != NULL && aux->valor < x) {
        ant = aux;
        aux = aux->prox;
    }
    if (aux == NULL || aux->valor != x) return l;
    if (ant == NULL) l = aux->prox;
    else ant->prox = aux->prox;
    free(aux);
    return l;
}
```

Remoção ordenada



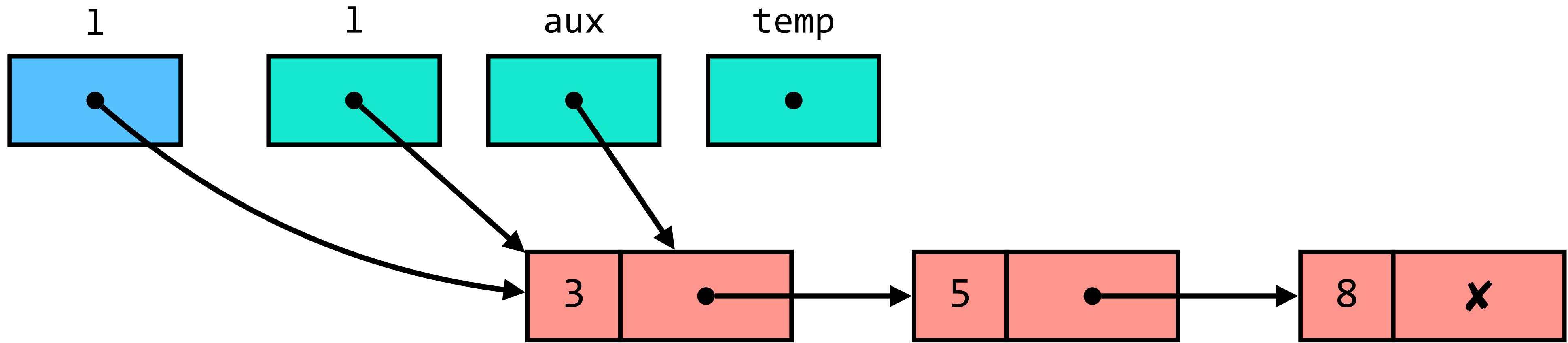
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = delete(5,l);
    return 0;
}
```

Remoção ordenada



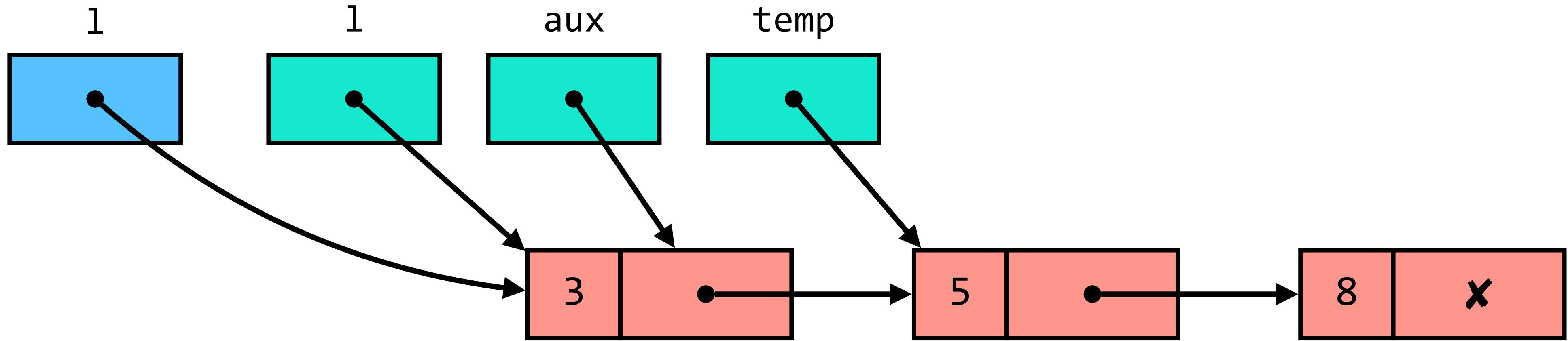
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = delete(5,l);
    return 0;
}
```

Remoção ordenada



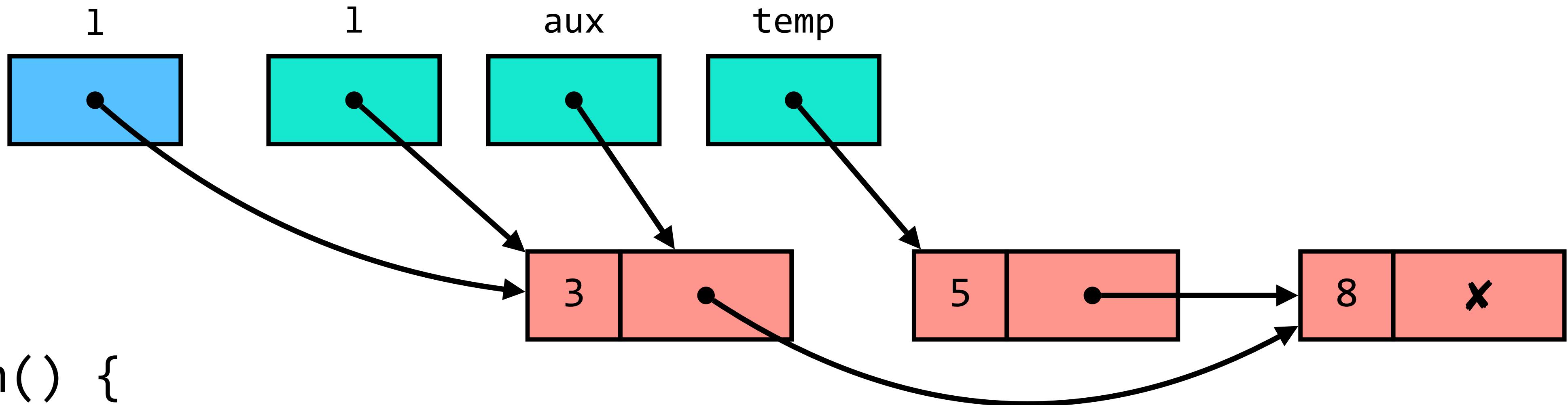
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = delete(5,l);
    return 0;
}
```

Remoção ordenada



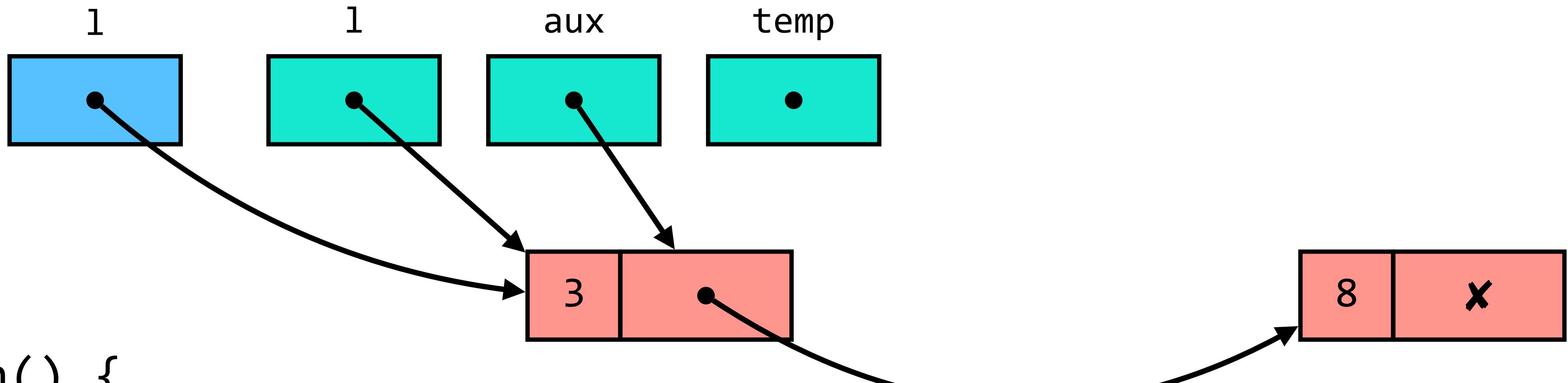
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = delete(5,l);
    return 0;
}
```

Remoção ordenada



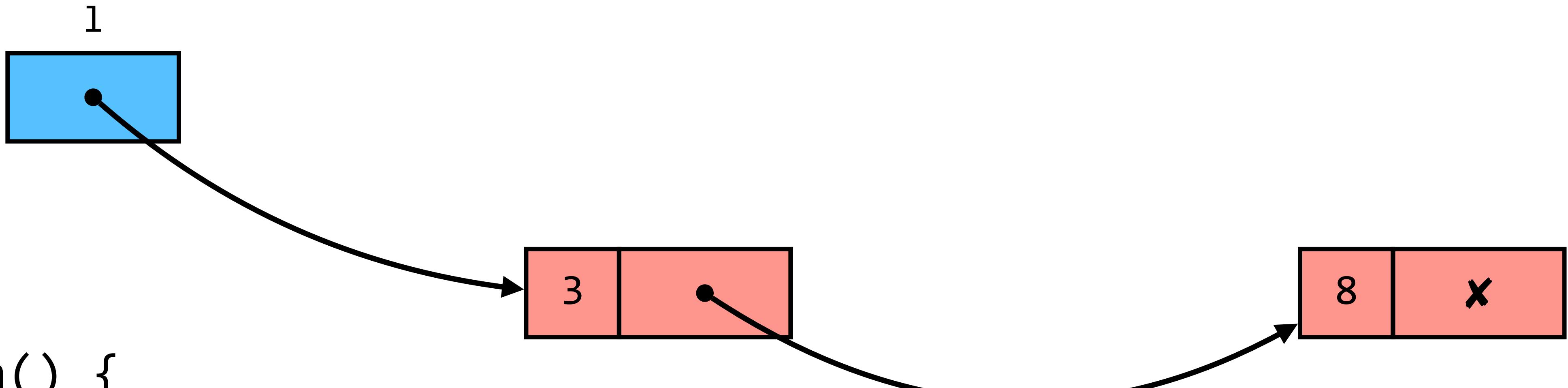
```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = delete(5,l);
    return 0;
}
```

Remoção ordenada



```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = delete(5,l);
    return 0;
}
```

Remoção ordenada



```
int main() {
    int v[] = {3,5,8};
    lint l = fromArray(v,3);
    l = delete(5,l);
    return 0;
}
```

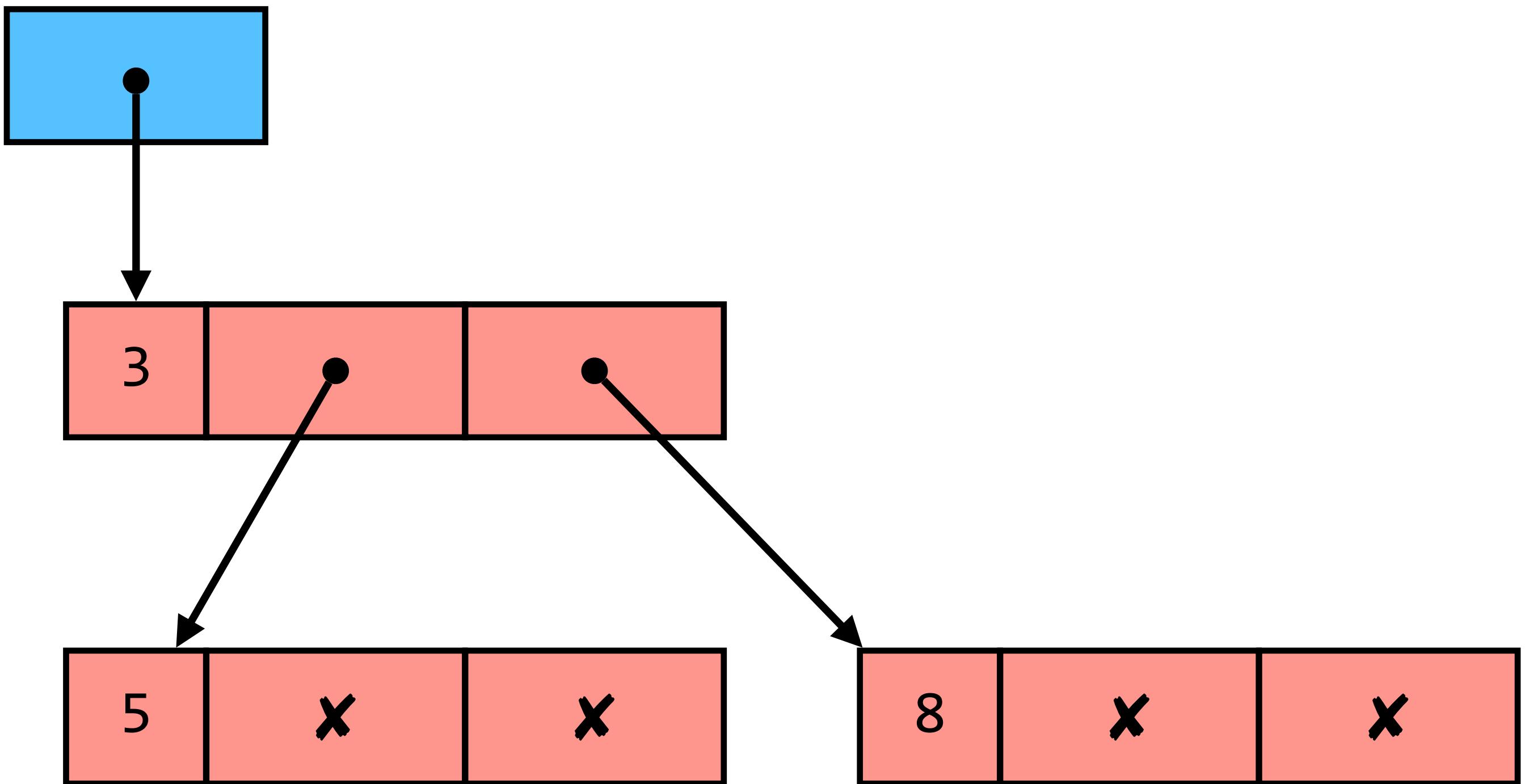
Remoção ordenada

```
lint delete(int x, lint l) {
    lint *aux = &l, temp;
    while ((*aux) != NULL && (*aux)->valor < x) {
        aux = &((*aux)->prox);
    }
    if ((*aux) == NULL || (*aux)->valor != x) return 1;
    temp = *aux;
    *aux = (*aux)->prox;
    free(temp);
    return 1;
}
```

Aula 13

Árvores binárias

```
typedef struct abin_no {  
    int valor;  
    struct abin_no *esq, *dir;  
} *abin;
```



Disclaimer

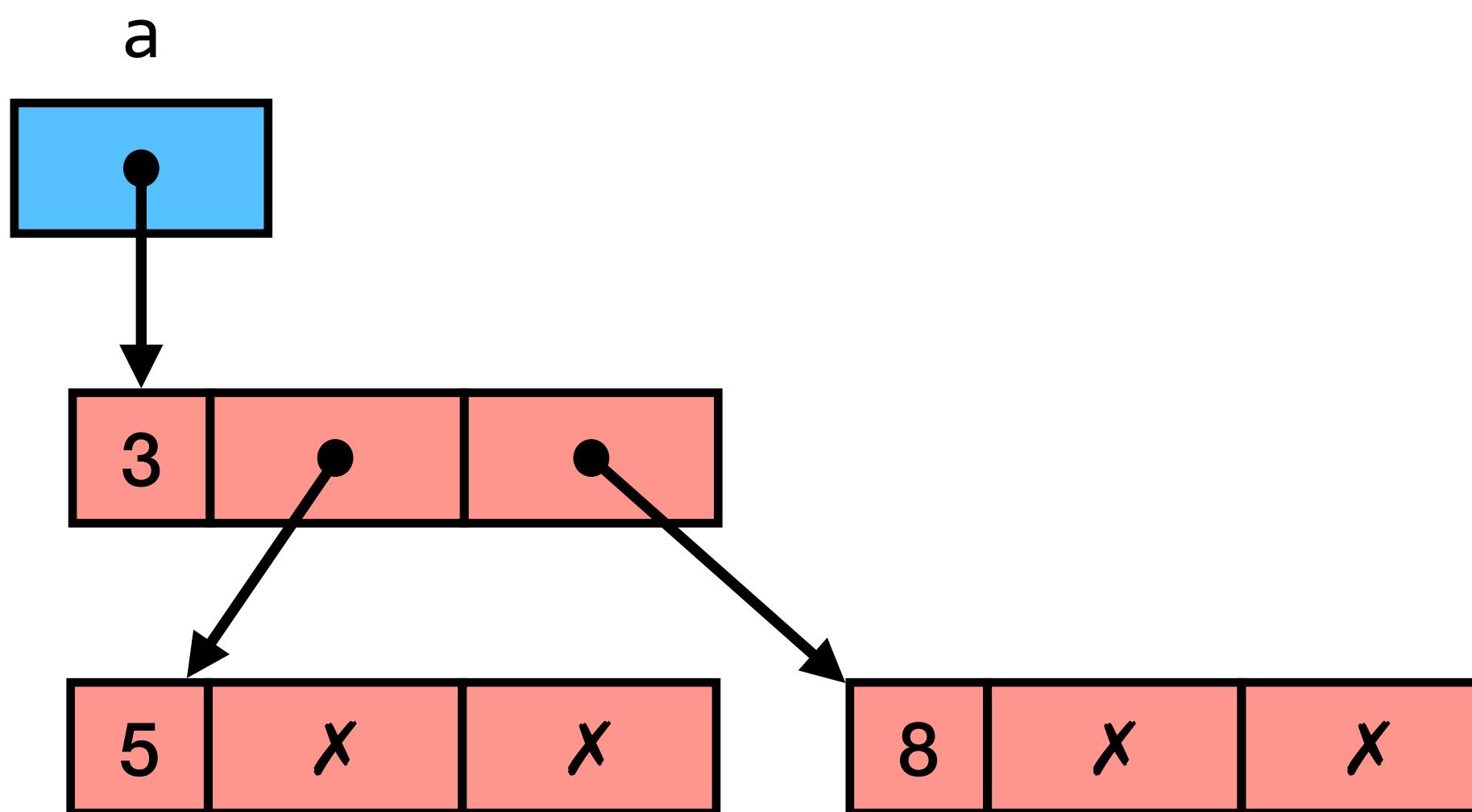


Inserção na raiz

```
abin mkroot(int x, abin e, abin d) {
    abin new = malloc(sizeof(struct abin_no));
    new->valor = x;
    new->esq = e;
    new->dir = d;
    return new;
}
```

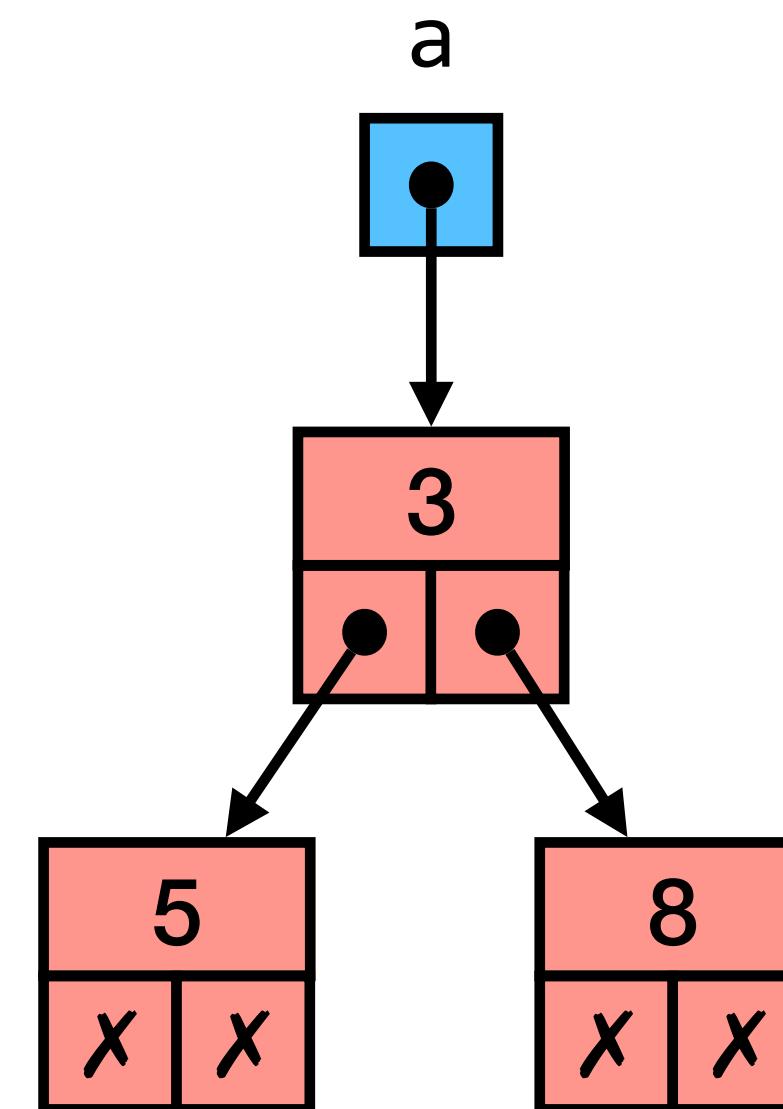
Inserção na raiz

```
int main() {
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));
    return 0;
}
```



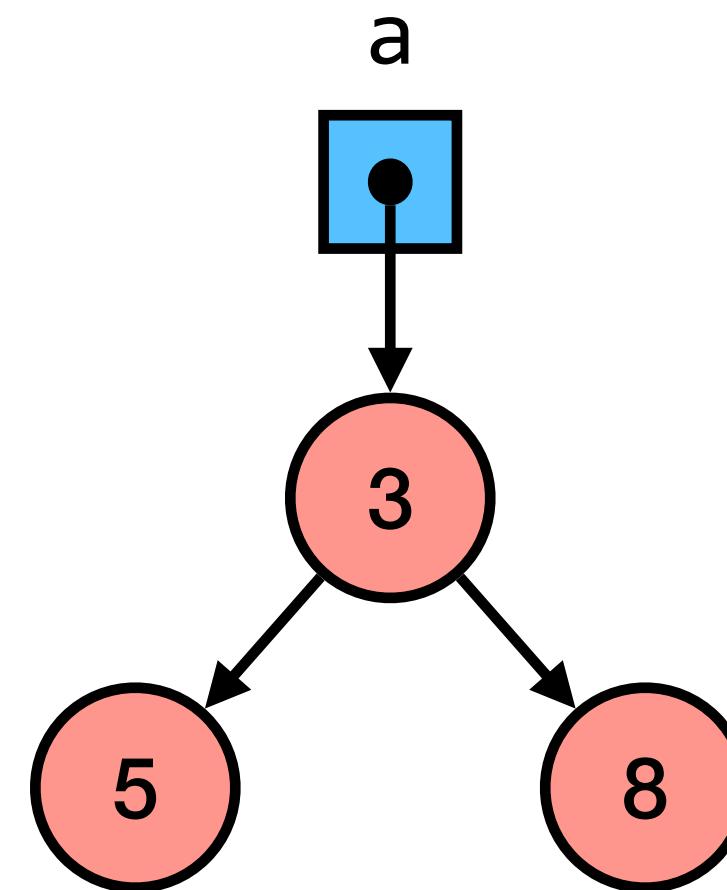
Inserção na raiz

```
int main() {
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));
    return 0;
}
```



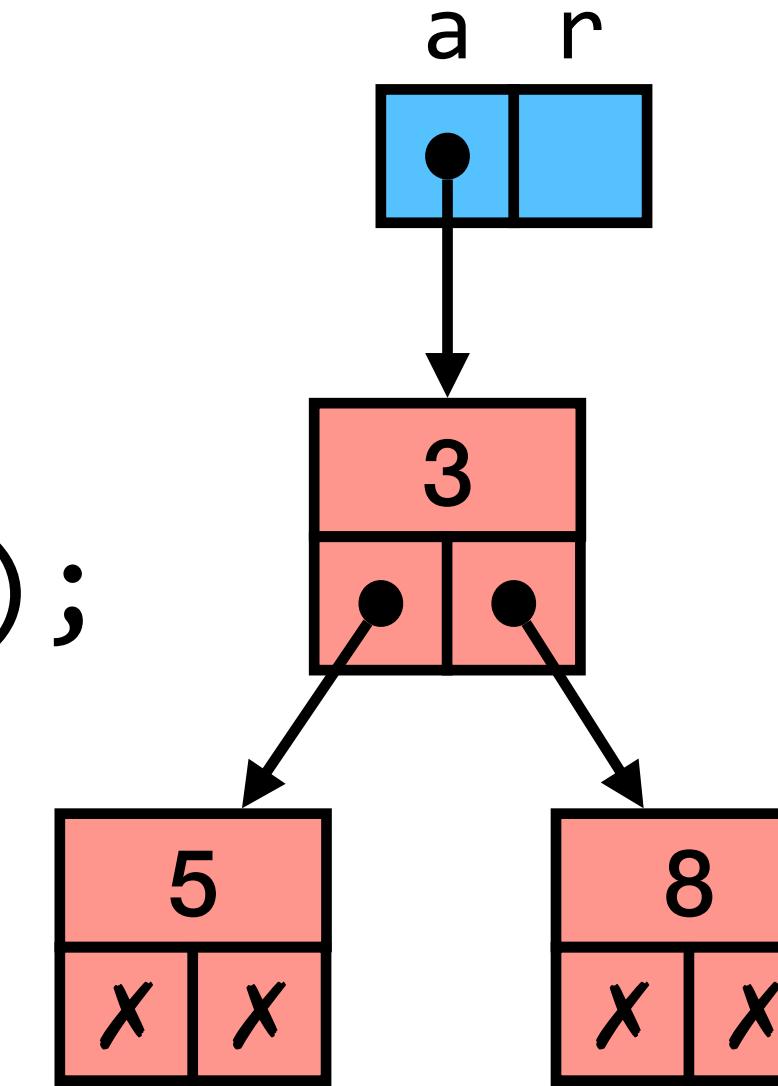
Inserção na raiz

```
int main() {
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));
    return 0;
}
```



Tamanho

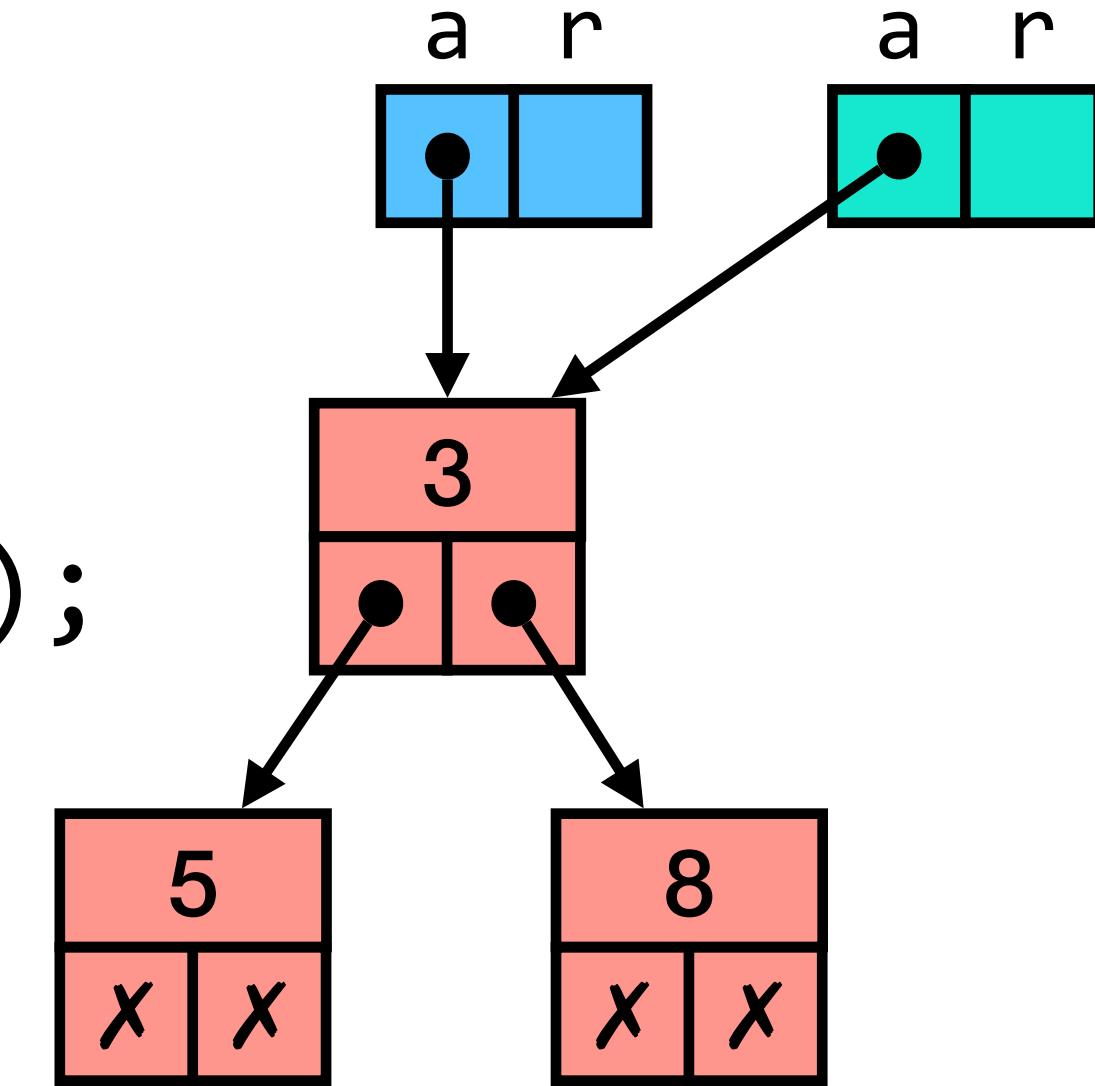
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```



```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Tamanho

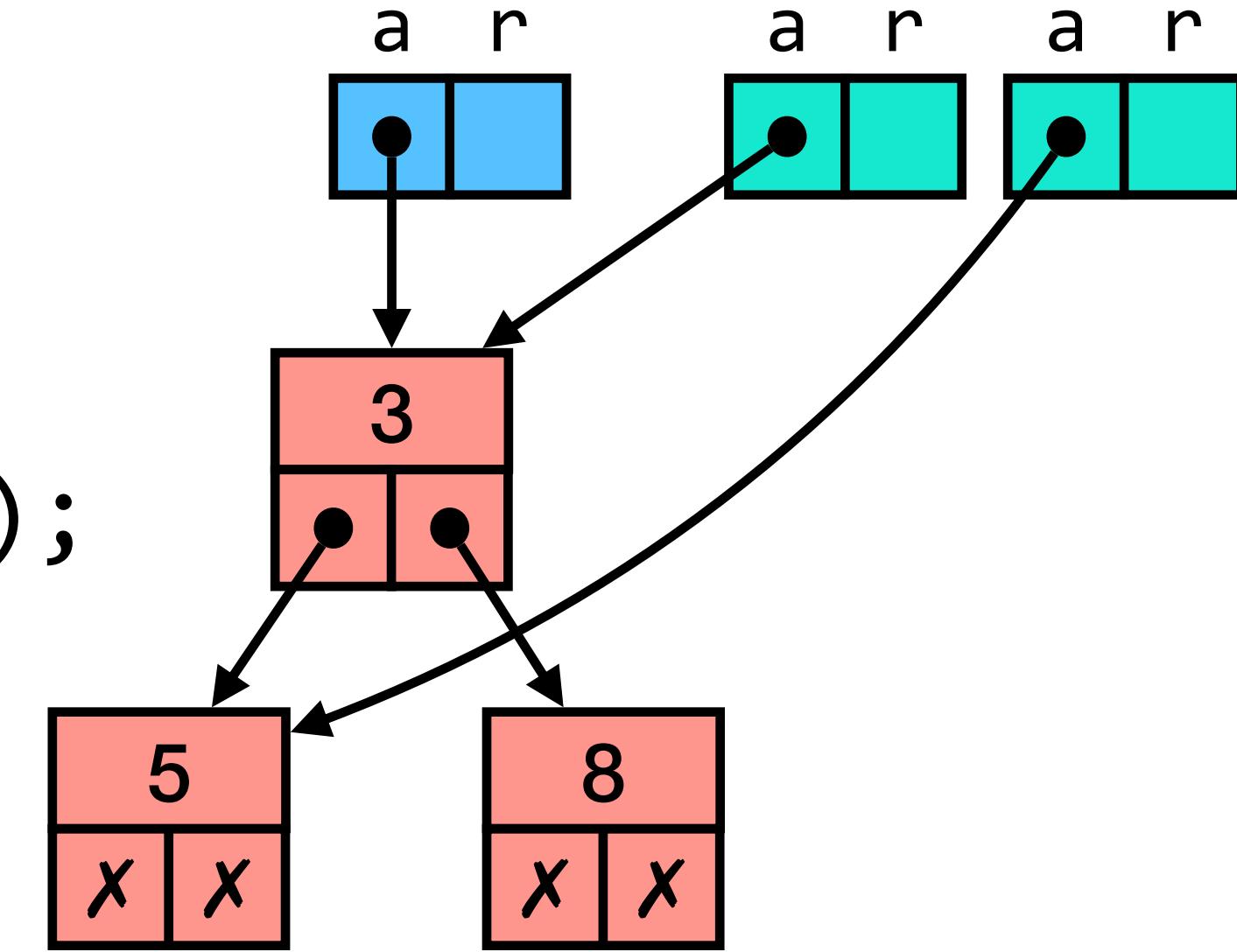
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```



```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Tamanho

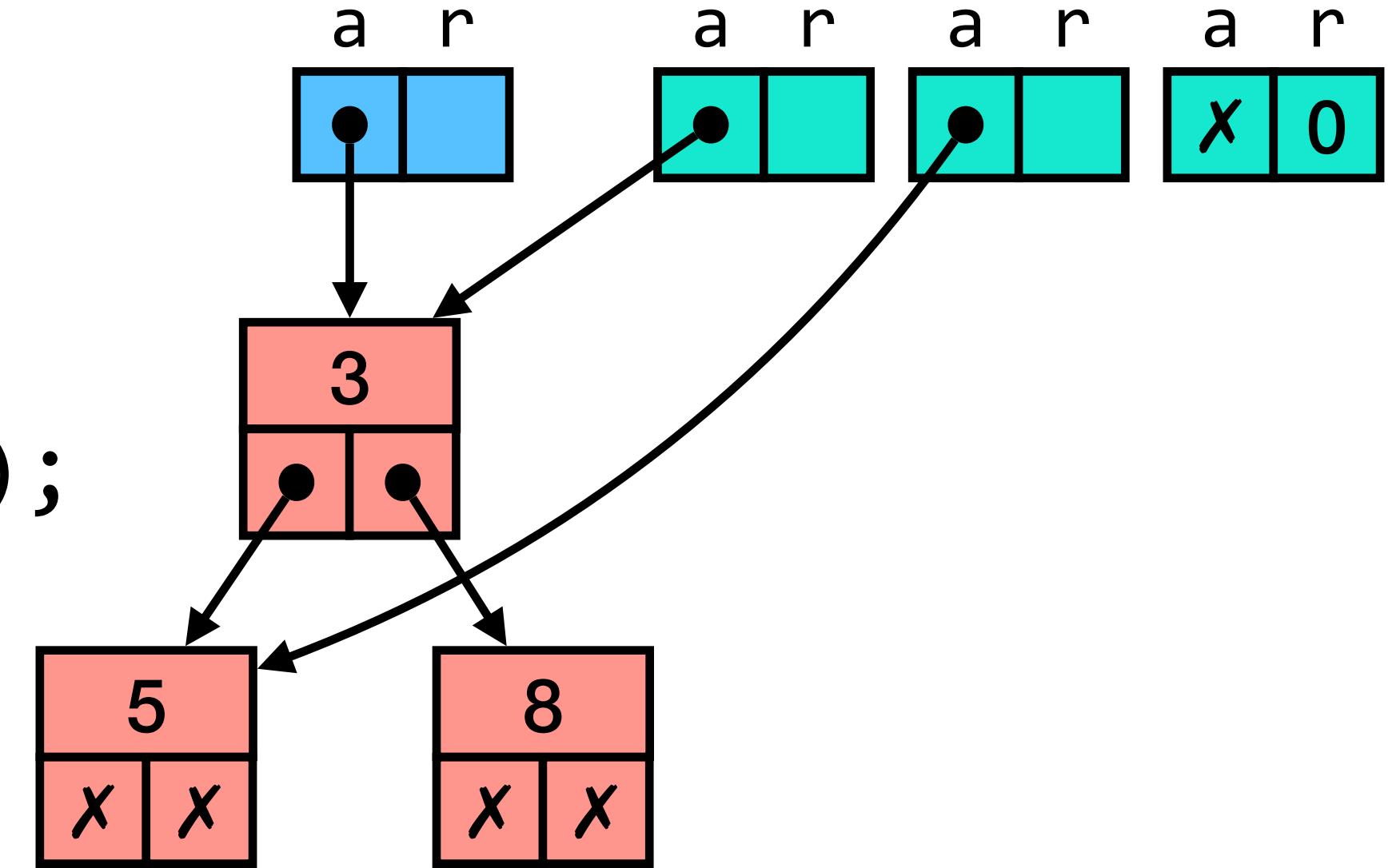
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```



```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Tamanho

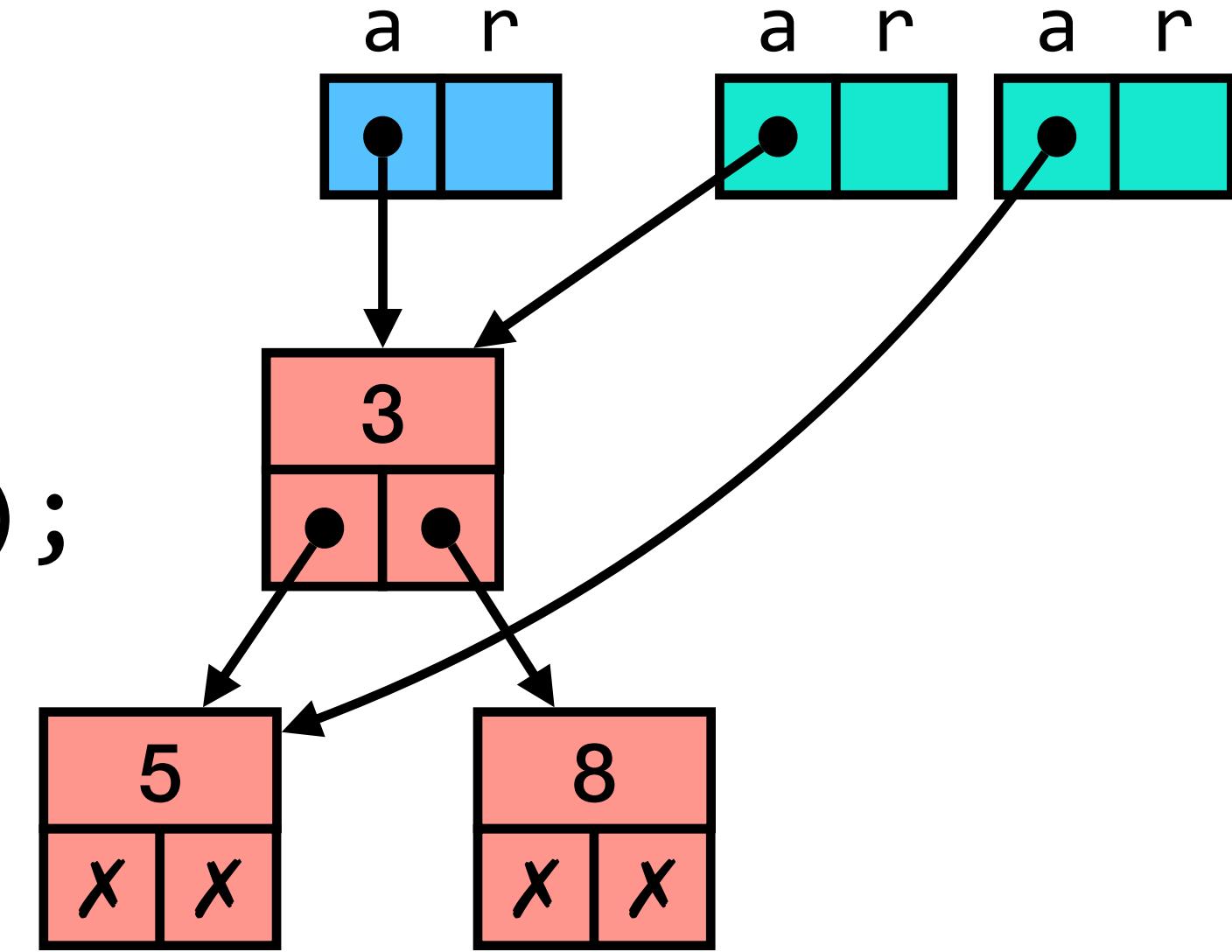
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```



```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Tamanho

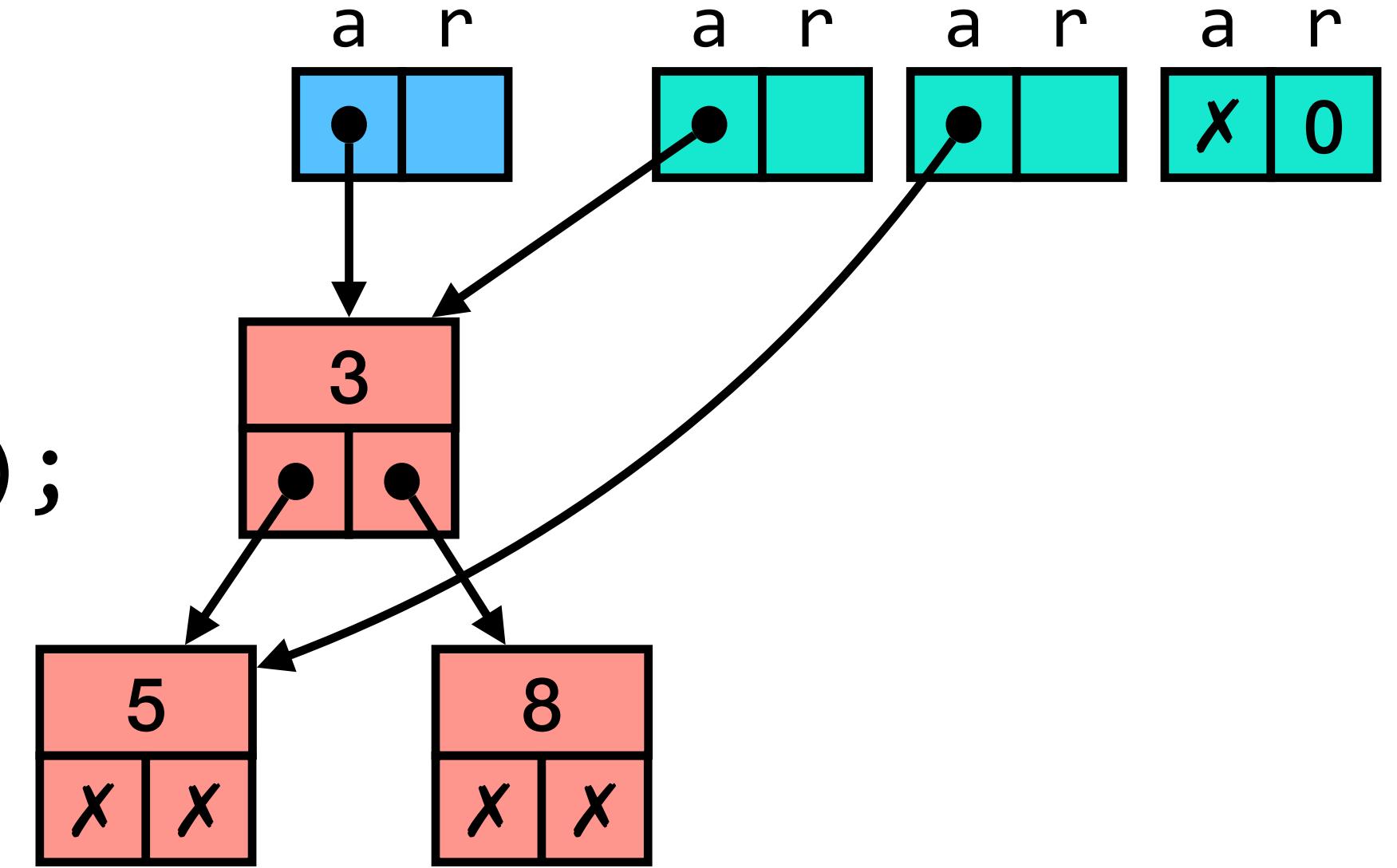
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```



```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Tamanho

```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```

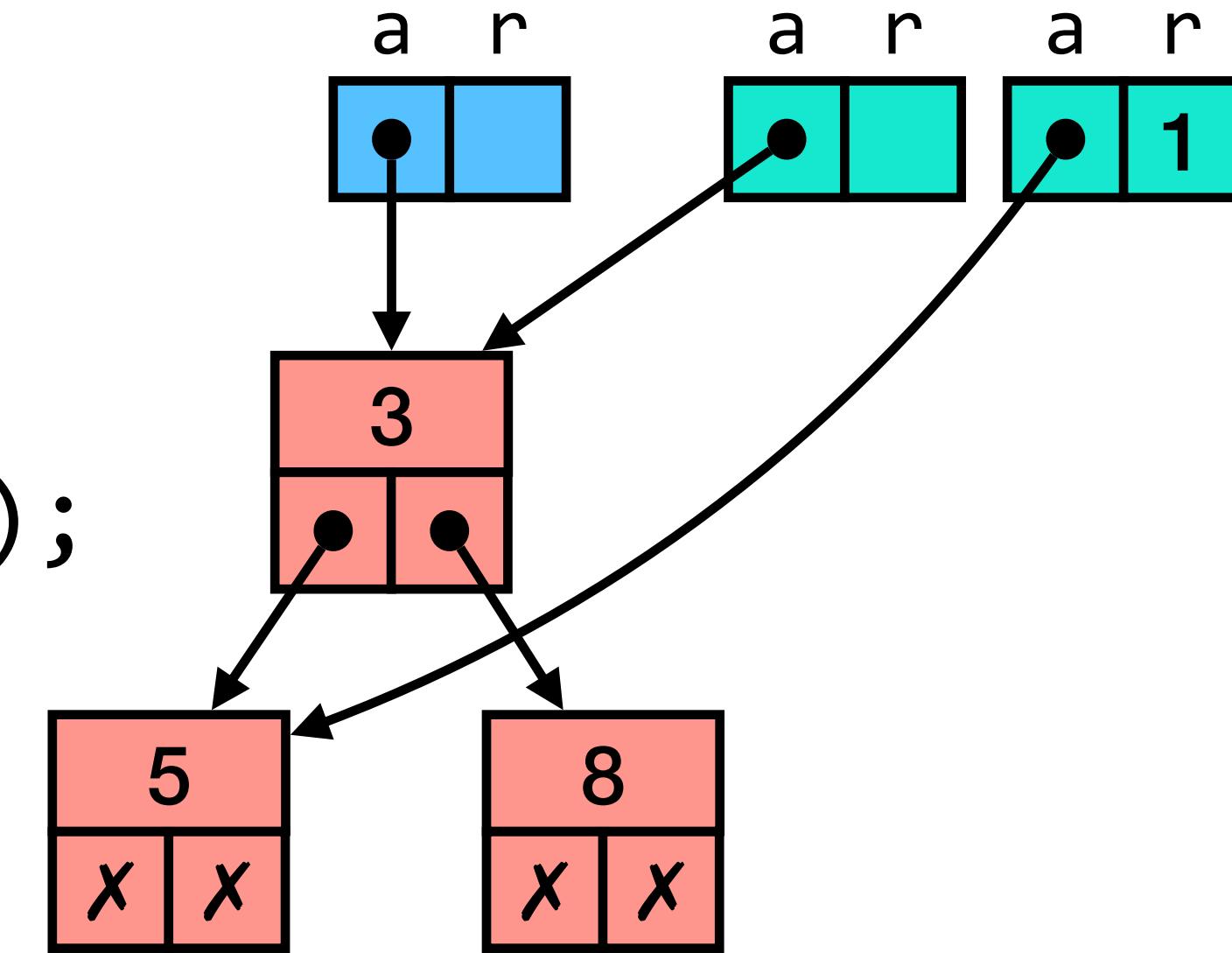


```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Tamanho

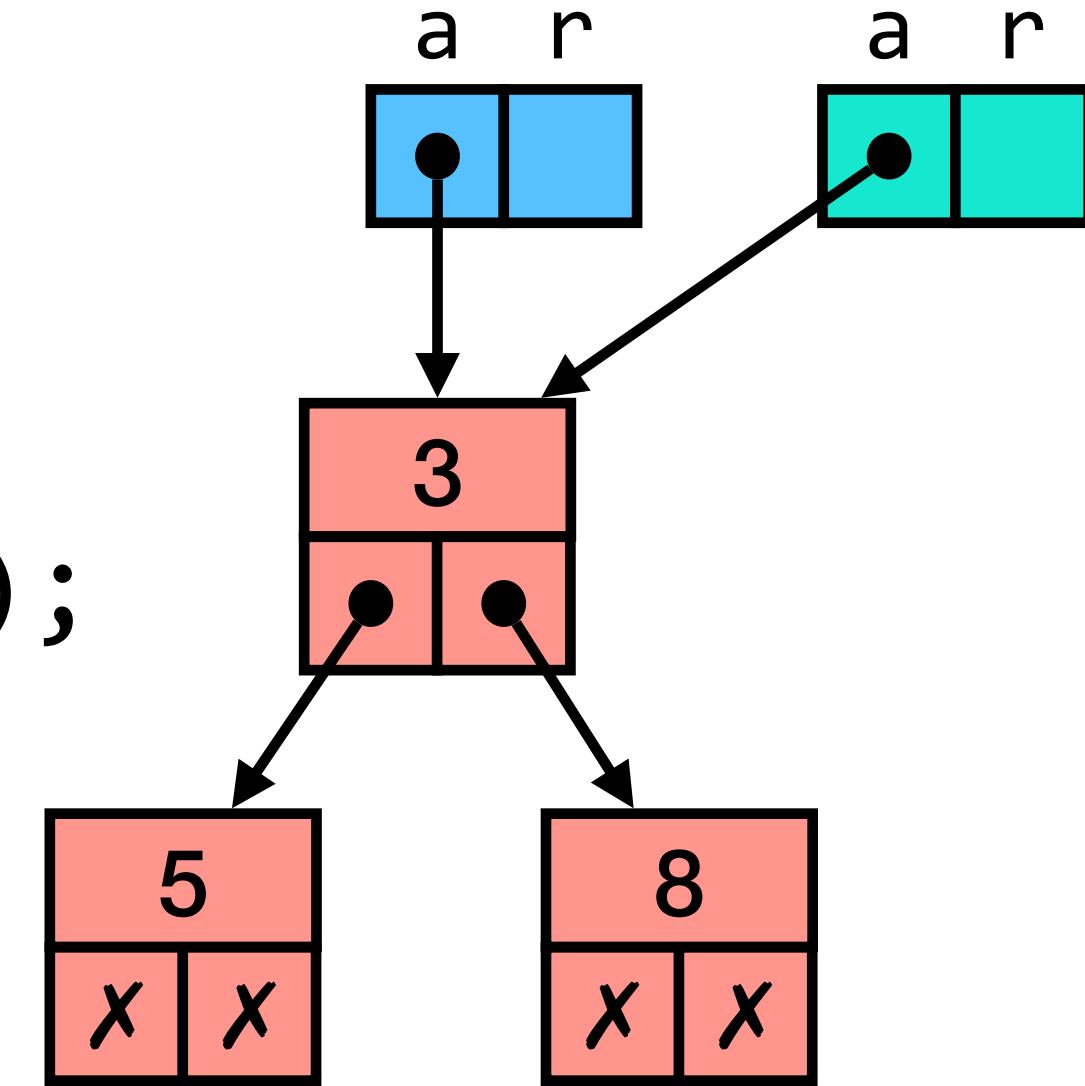
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```

```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```



Tamanho

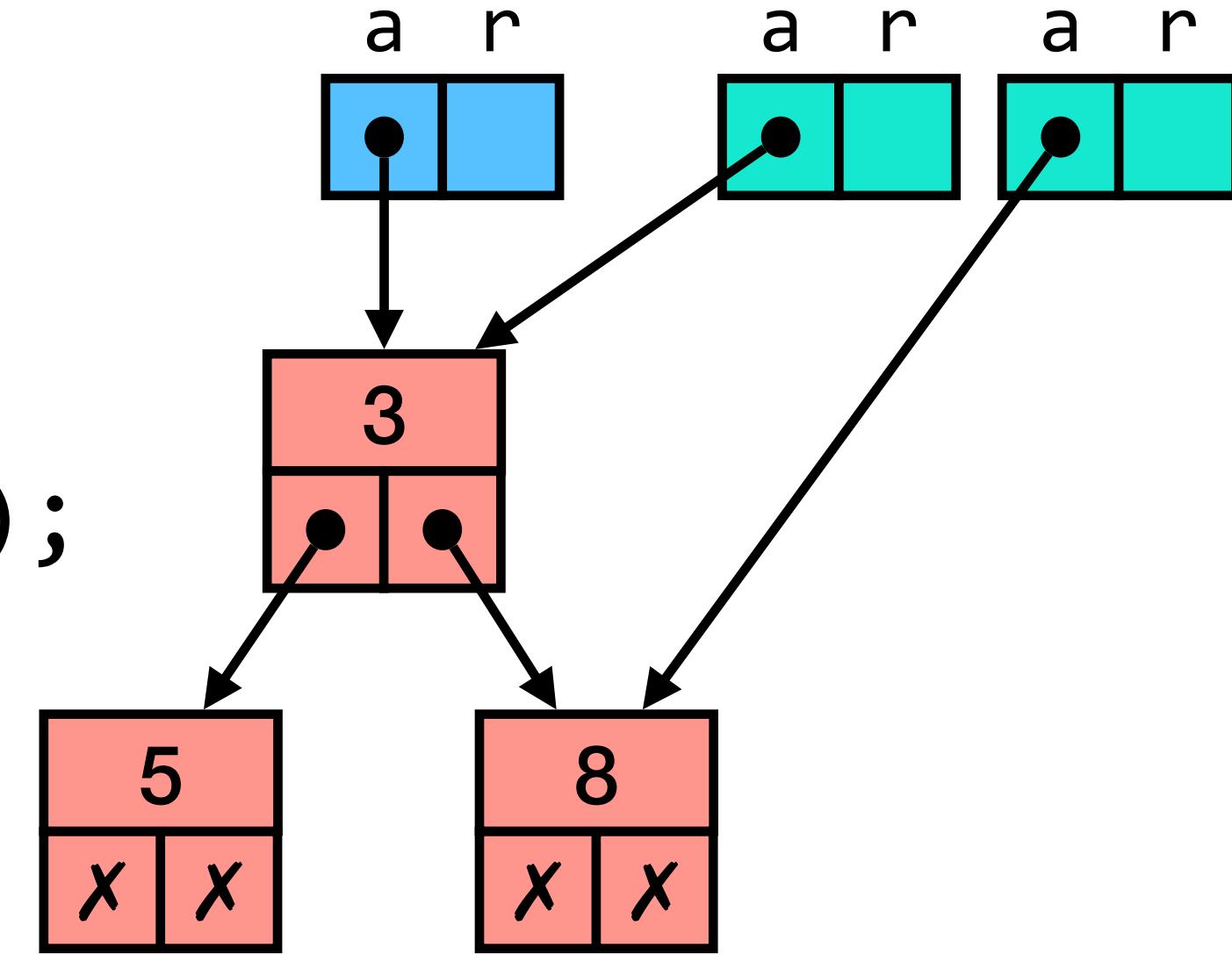
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```



```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Tamanho

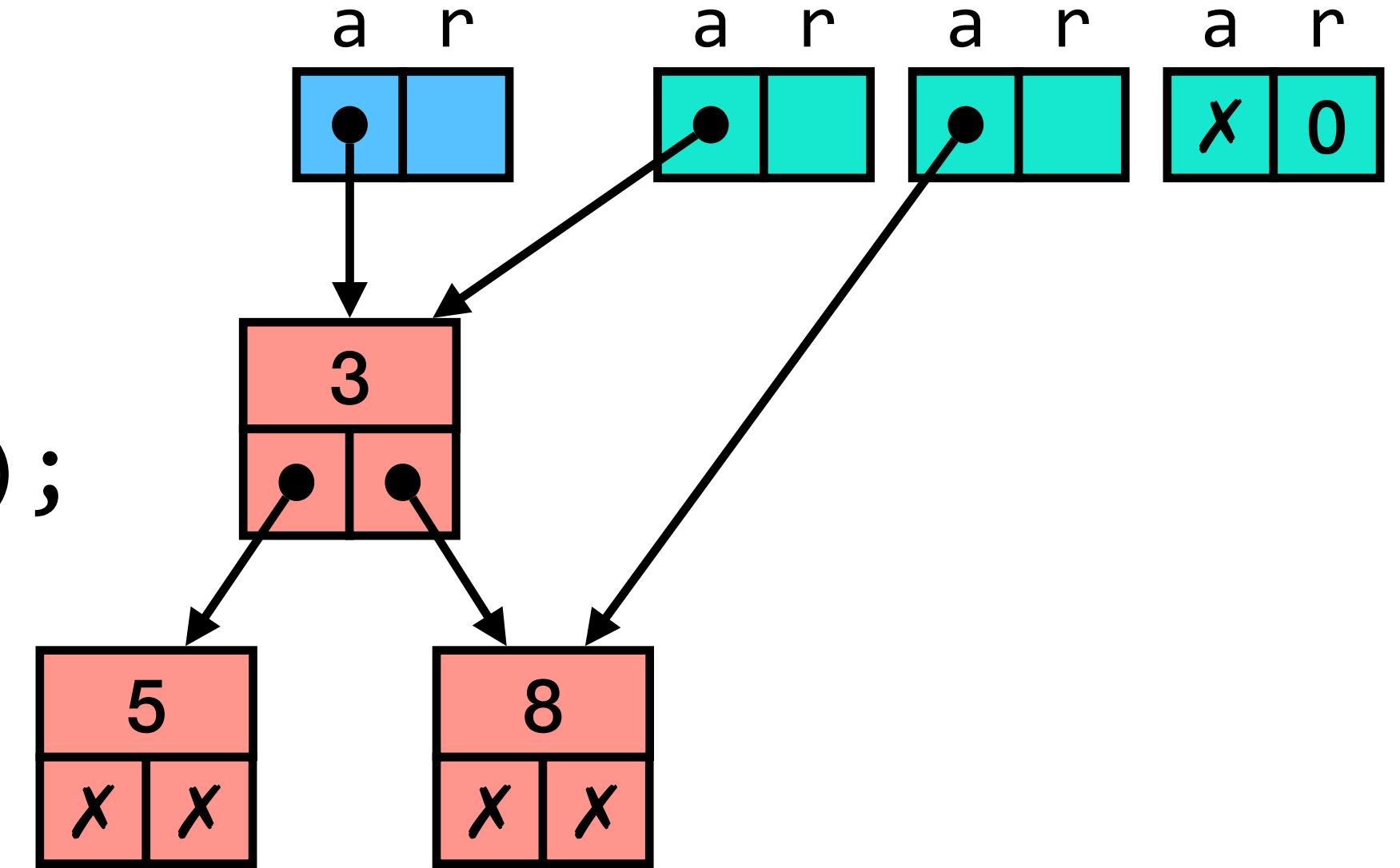
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```



```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Tamanho

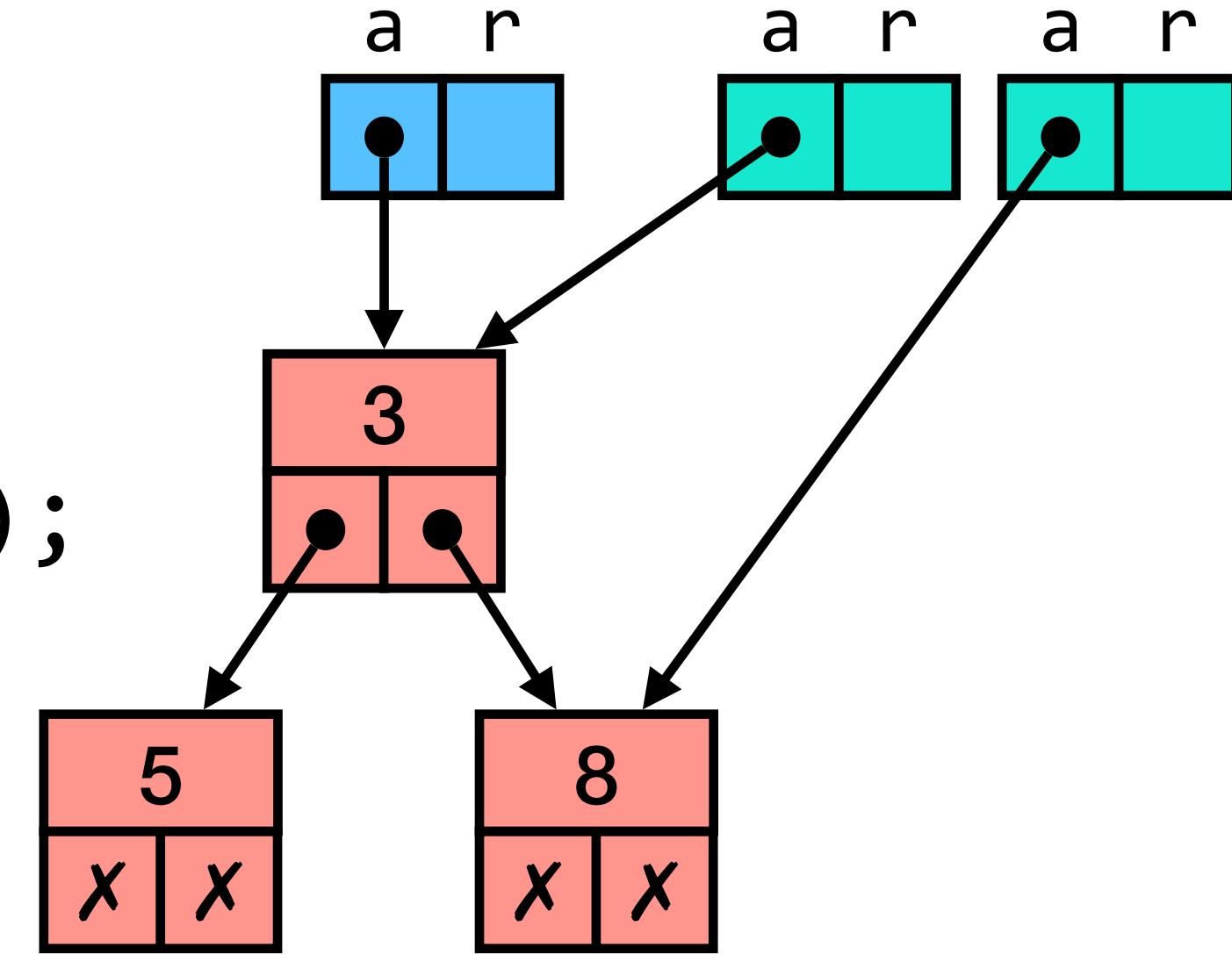
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```



```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Tamanho

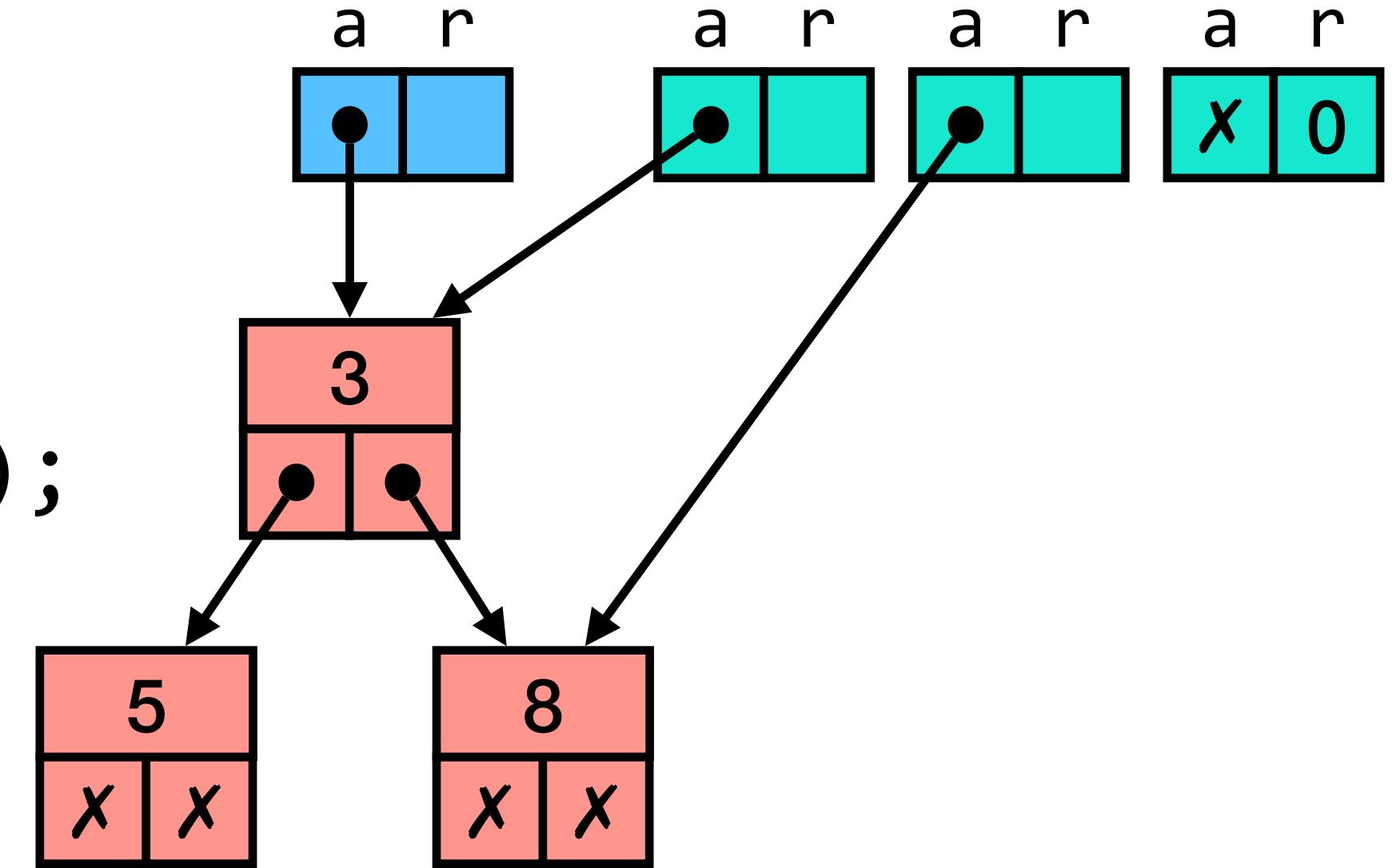
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```



```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Tamanho

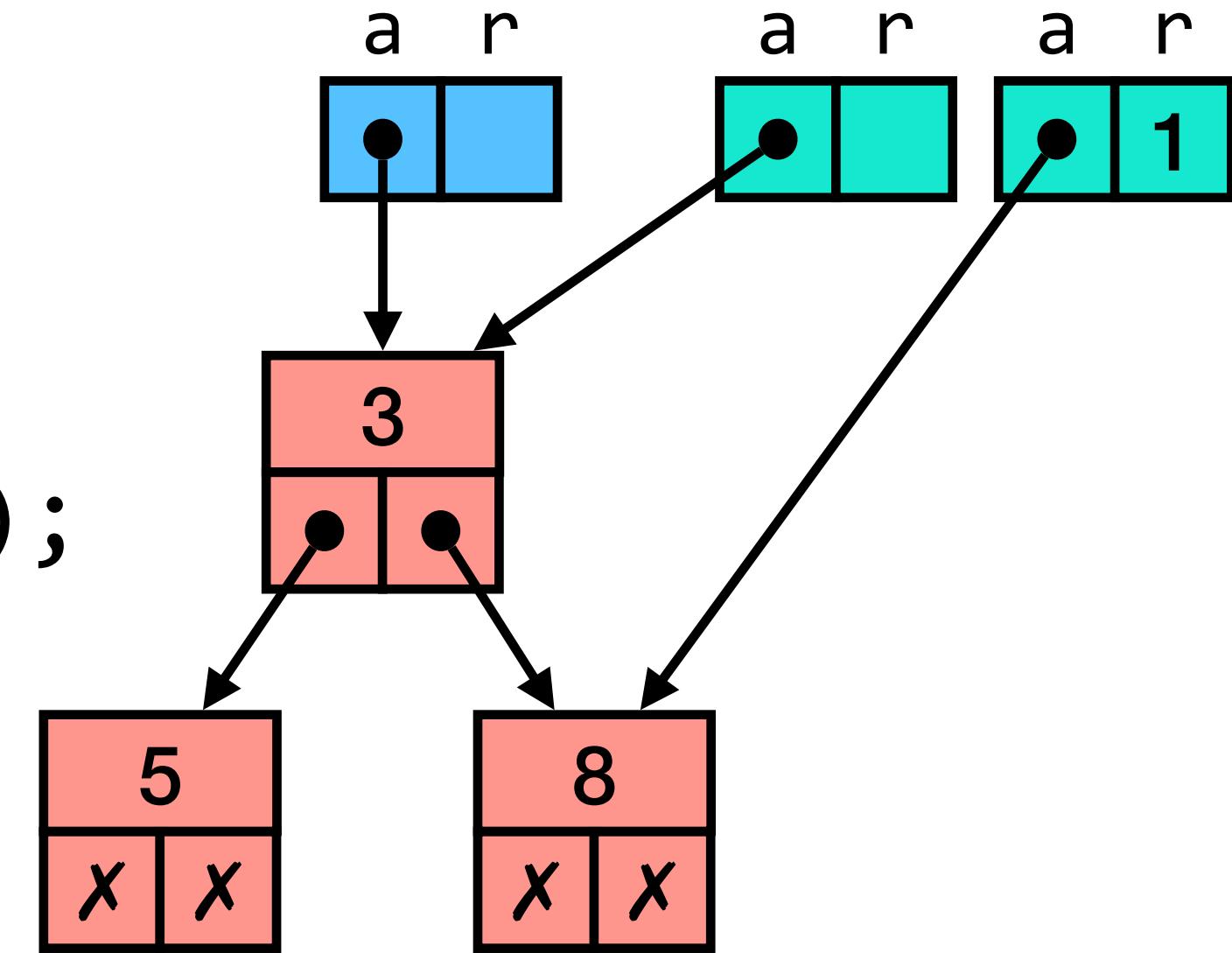
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```



```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Tamanho

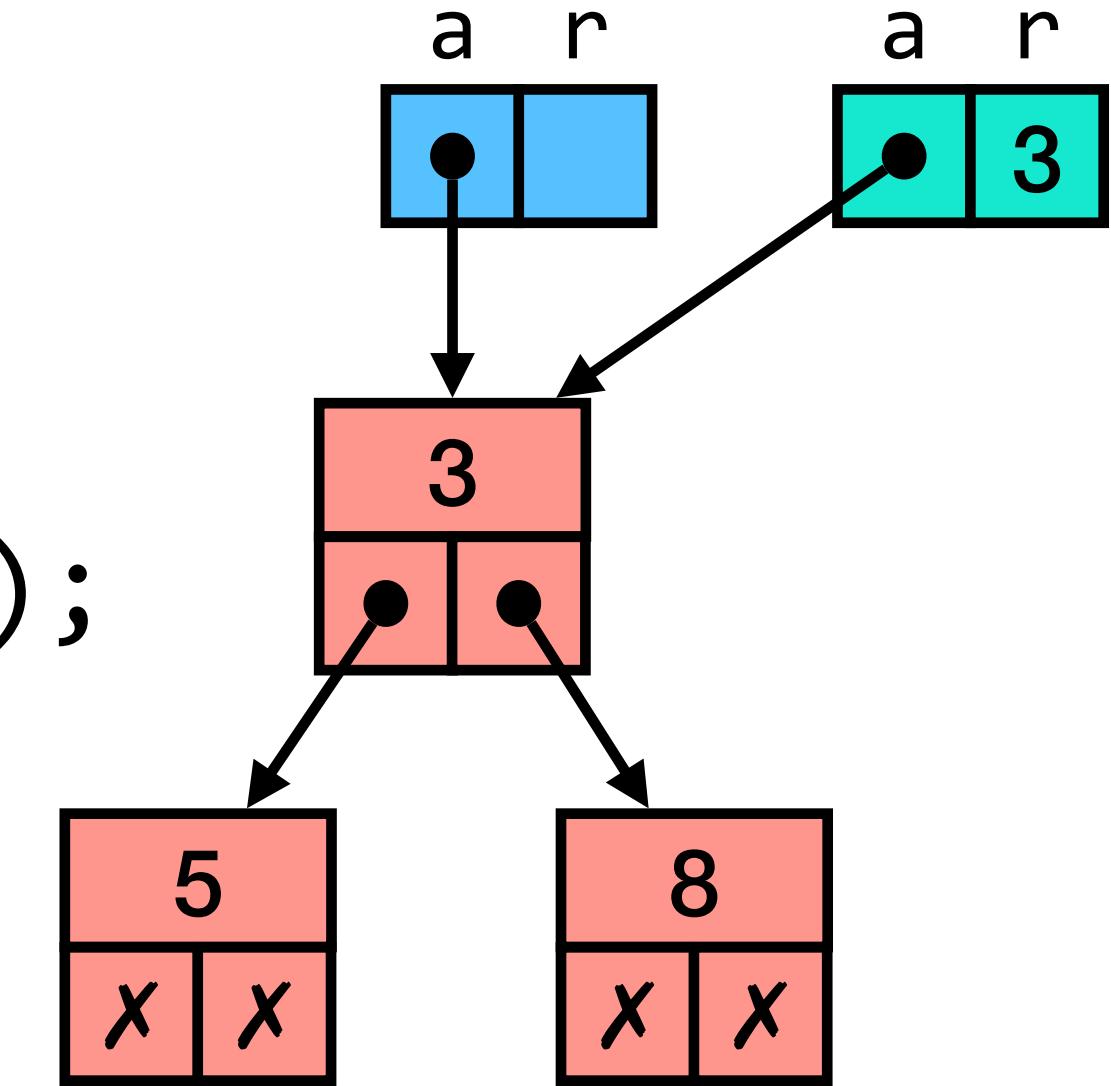
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```



```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Tamanho

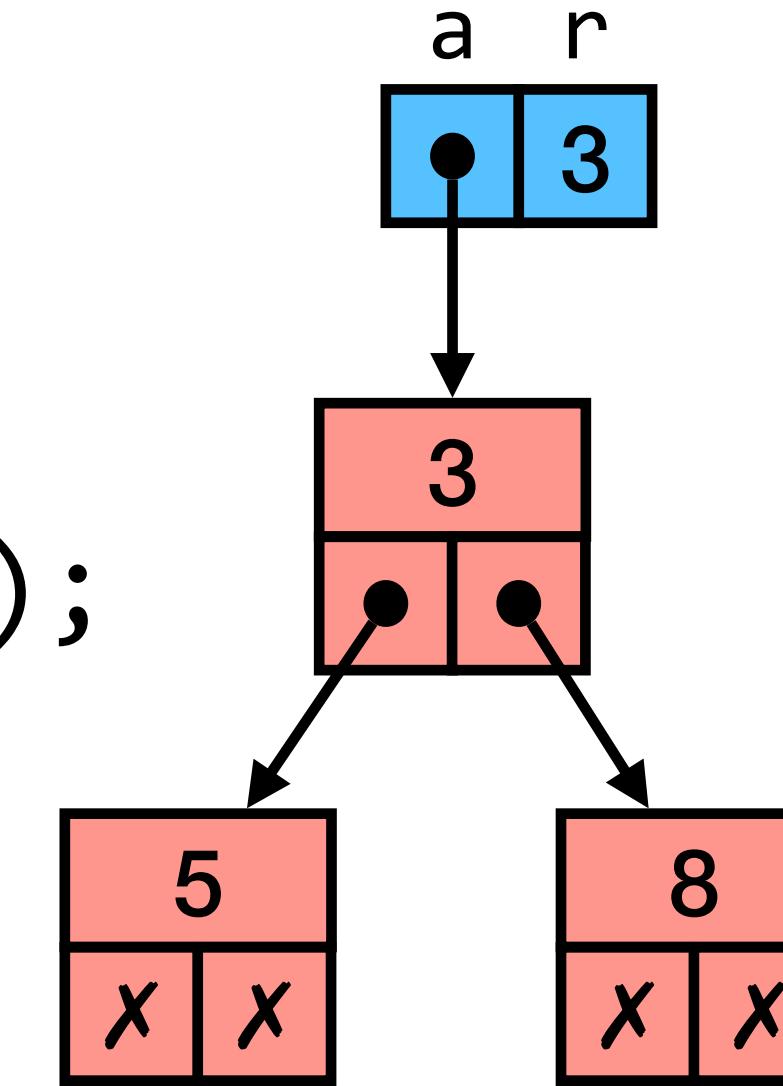
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```



```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Tamanho

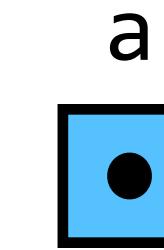
```
int size(abin a) {  
    int r;  
    if (a == NULL) r = 0;  
    else r = 1 + size(a->esq) + size(a->dir);  
    return r;  
}
```



```
int main() {  
    abin a = mkroot(3, mkroot(5, NULL, NULL), mkroot(8, NULL, NULL));  
    int r = size(a);  
    return 0;  
}
```

Criação a partir de array

```
abin fromArray(int v[], int N) {  
    abin a = NULL;  
    for (int i = 0; i < N; i++) a = mkroot(v[i], a, NULL);  
    return a;  
}
```

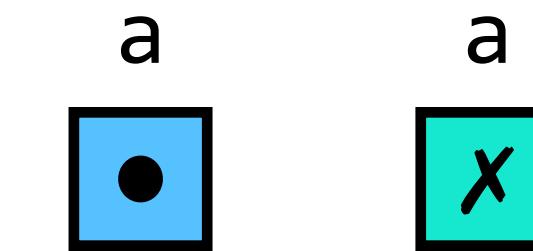


```
int main() {  
    int v[] = {5,3,8};  
    abin a = fromArray(v, 3);  
    return 0;  
}
```

Criação a partir de array

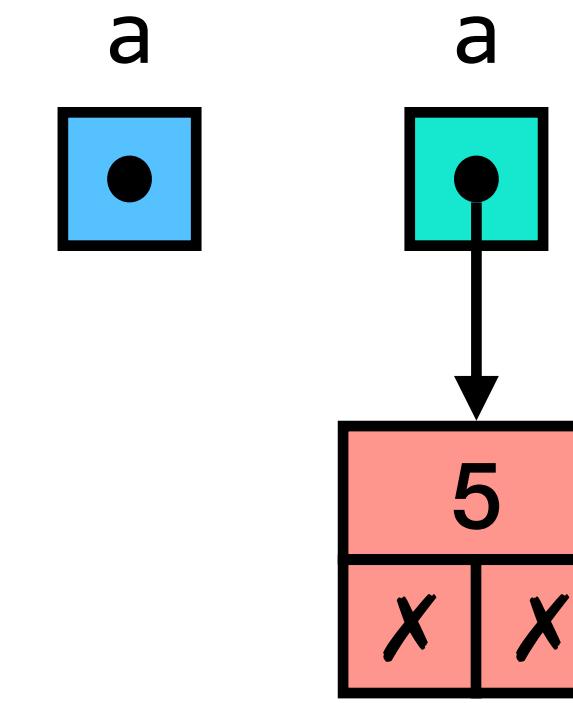
```
abin fromArray(int v[], int N) {  
    abin a = NULL;  
    for (int i = 0; i < N; i++) a = mkroot(v[i], a, NULL);  
    return a;  
}
```

```
int main() {  
    int v[] = {5,3,8};  
    abin a = fromArray(v, 3);  
    return 0;  
}
```



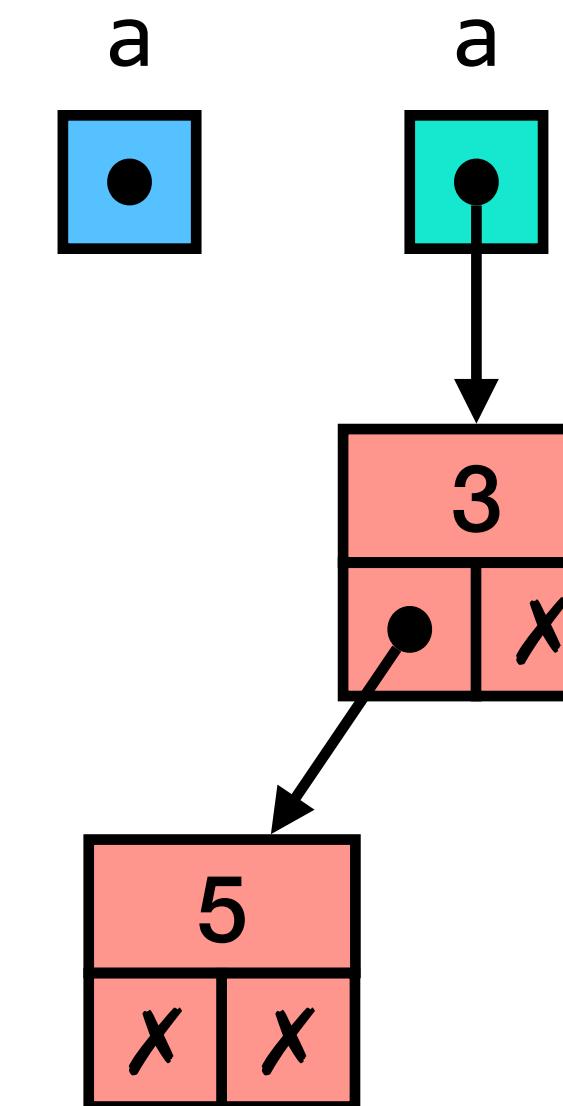
Criação a partir de array

```
abin fromArray(int v[], int N) {  
    abin a = NULL;  
    for (int i = 0; i < N; i++) a = mkroot(v[i], a, NULL);  
    return a;  
}  
  
int main() {  
    int v[] = {5,3,8};  
    abin a = fromArray(v, 3);  
    return 0;  
}
```



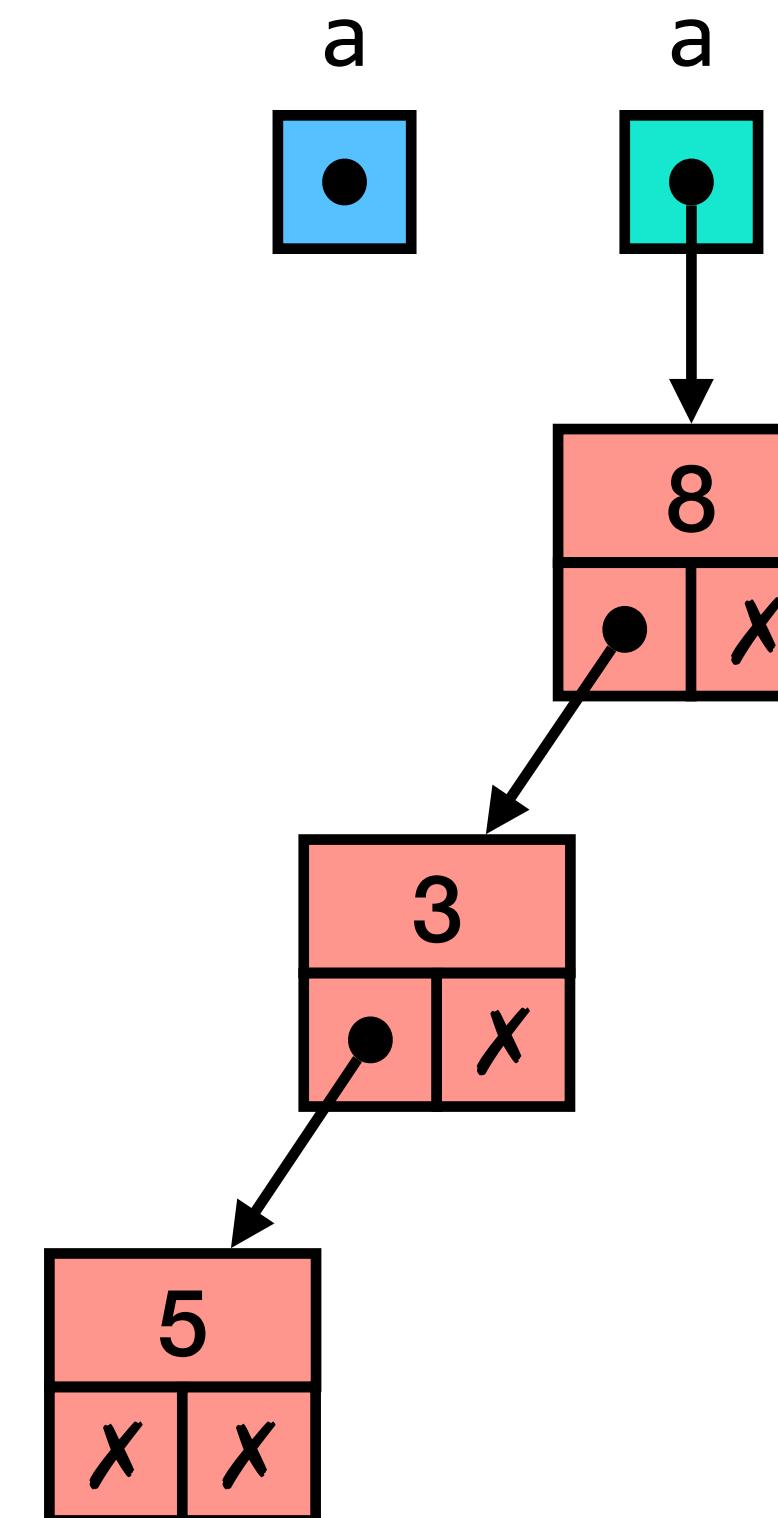
Criação a partir de array

```
abin fromArray(int v[], int N) {  
    abin a = NULL;  
    for (int i = 0; i < N; i++) a = mkroot(v[i], a, NULL);  
    return a;  
}  
  
int main() {  
    int v[] = {5,3,8};  
    abin a = fromArray(v, 3);  
    return 0;  
}
```



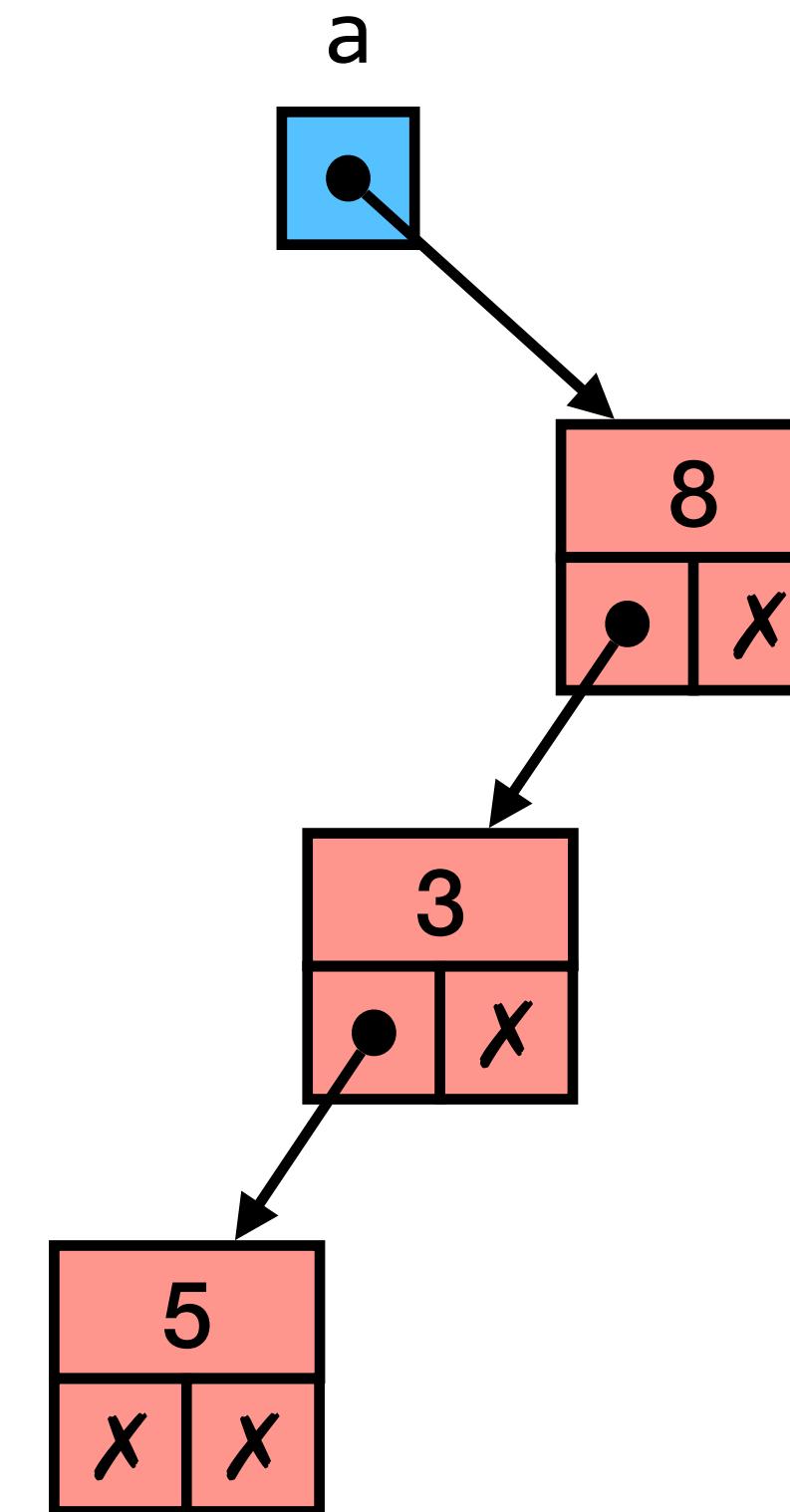
Criação a partir de array

```
abin fromArray(int v[], int N) {  
    abin a = NULL;  
    for (int i = 0; i < N; i++) a = mkroot(v[i], a, NULL);  
    return a;  
}  
  
int main() {  
    int v[] = {5,3,8};  
    abin a = fromArray(v, 3);  
    return 0;  
}
```



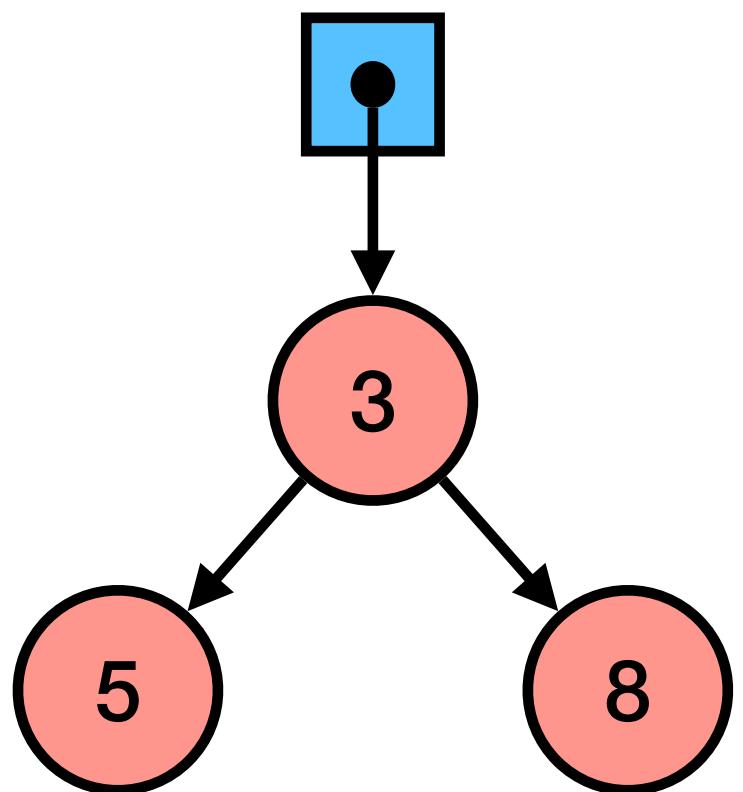
Criação a partir de array

```
abin fromArray(int v[], int N) {  
    abin a = NULL;  
    for (int i = 0; i < N; i++) a = mkroot(v[i], a, NULL);  
    return a;  
}  
  
int main() {  
    int v[] = {5,3,8};  
    abin a = fromArray(v, 3);  
    return 0;  
}
```

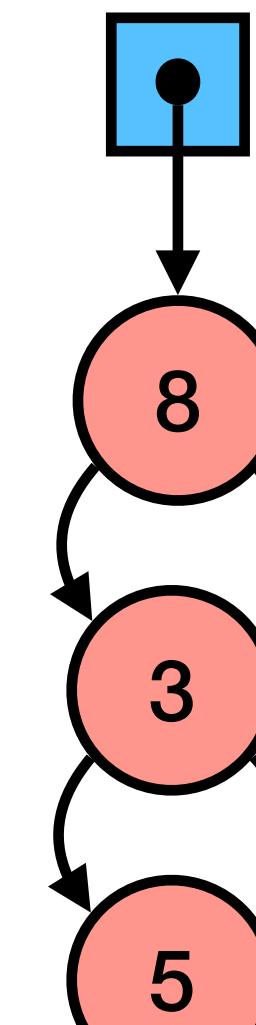


#26 Que árvore vai ser criada?

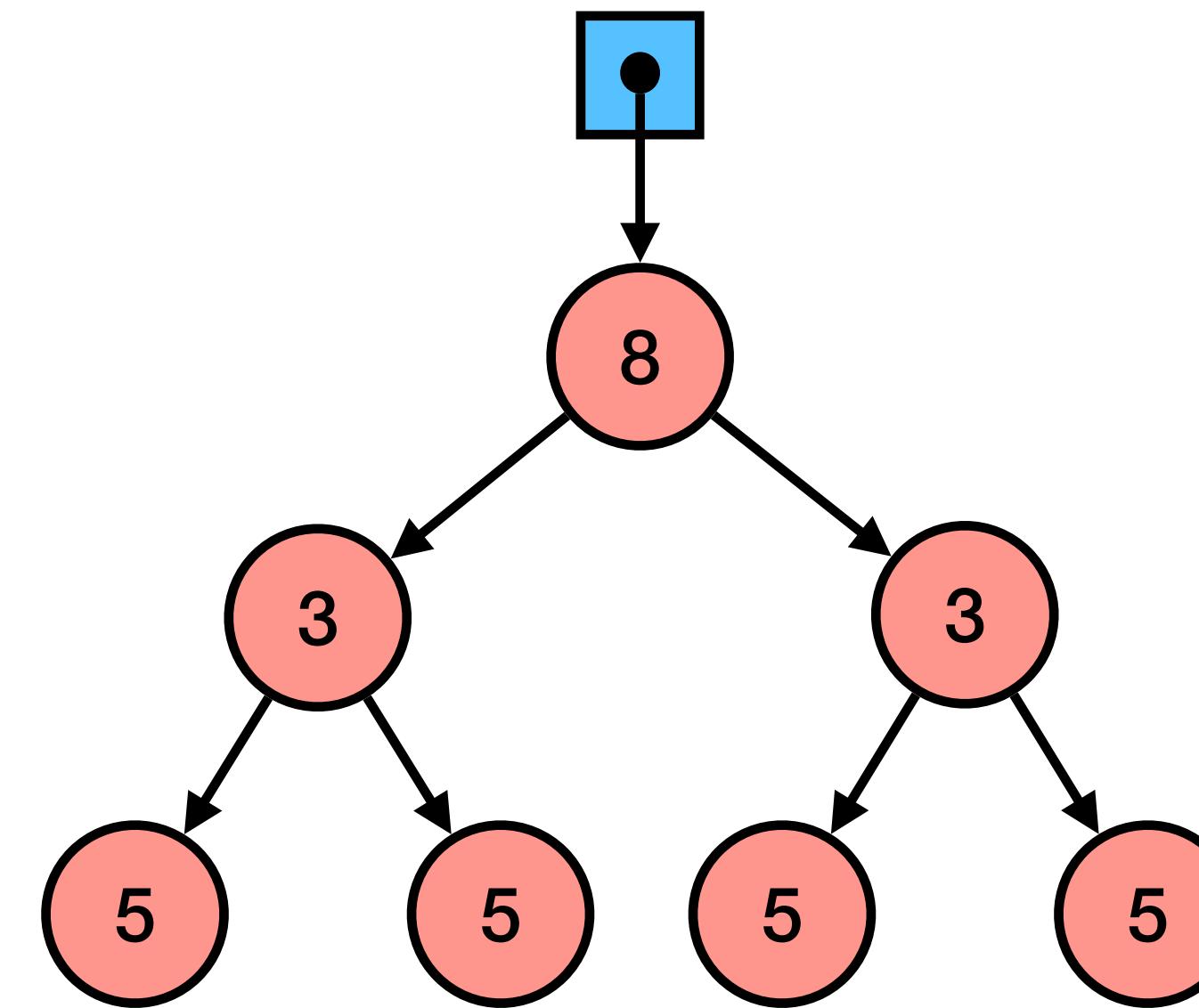
```
int v[ ] = {5,3,8}; abin a = NULL;  
for (int i = 0; i < N; i++) a = mkroot(v[i], a, a);
```



A



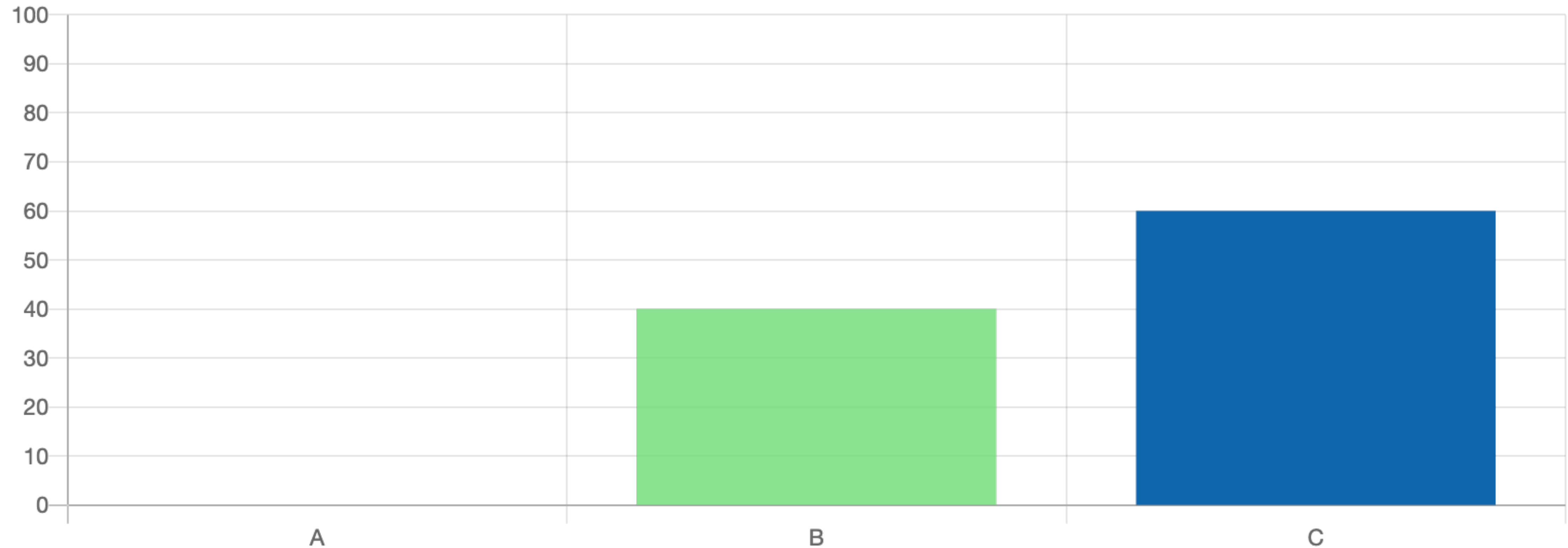
B



C



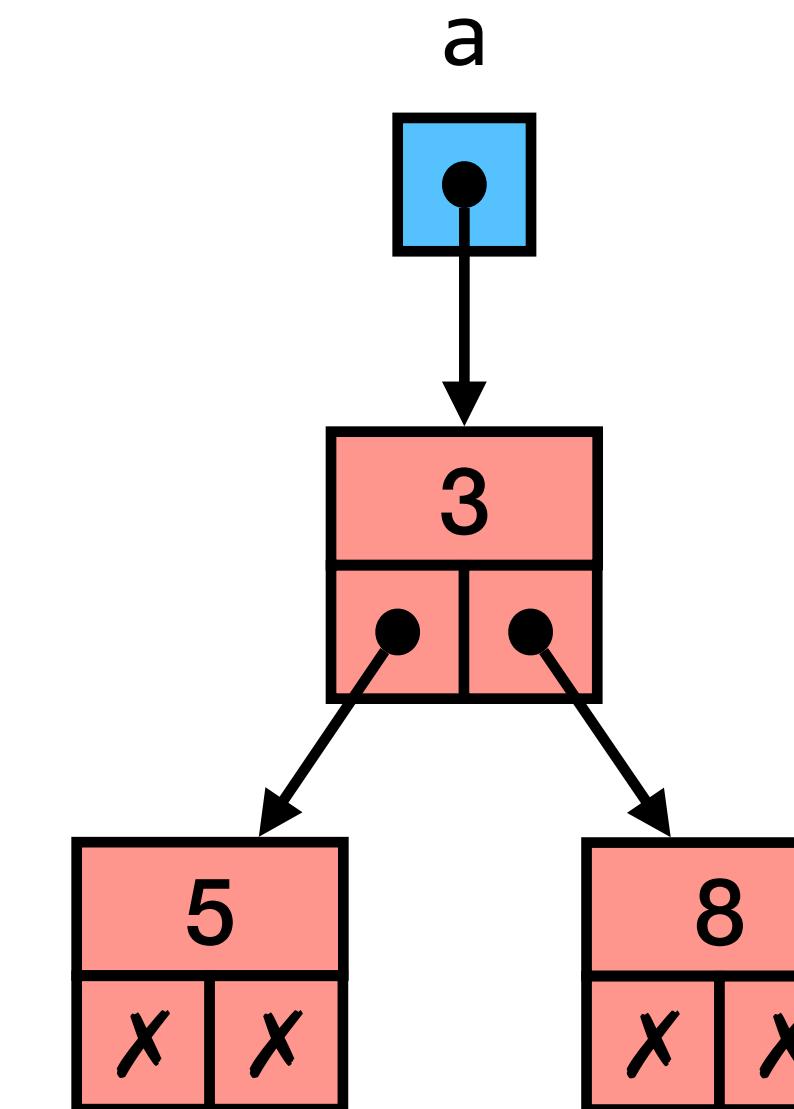
#26 Que árvore vai ser criada?



Criação a partir de array

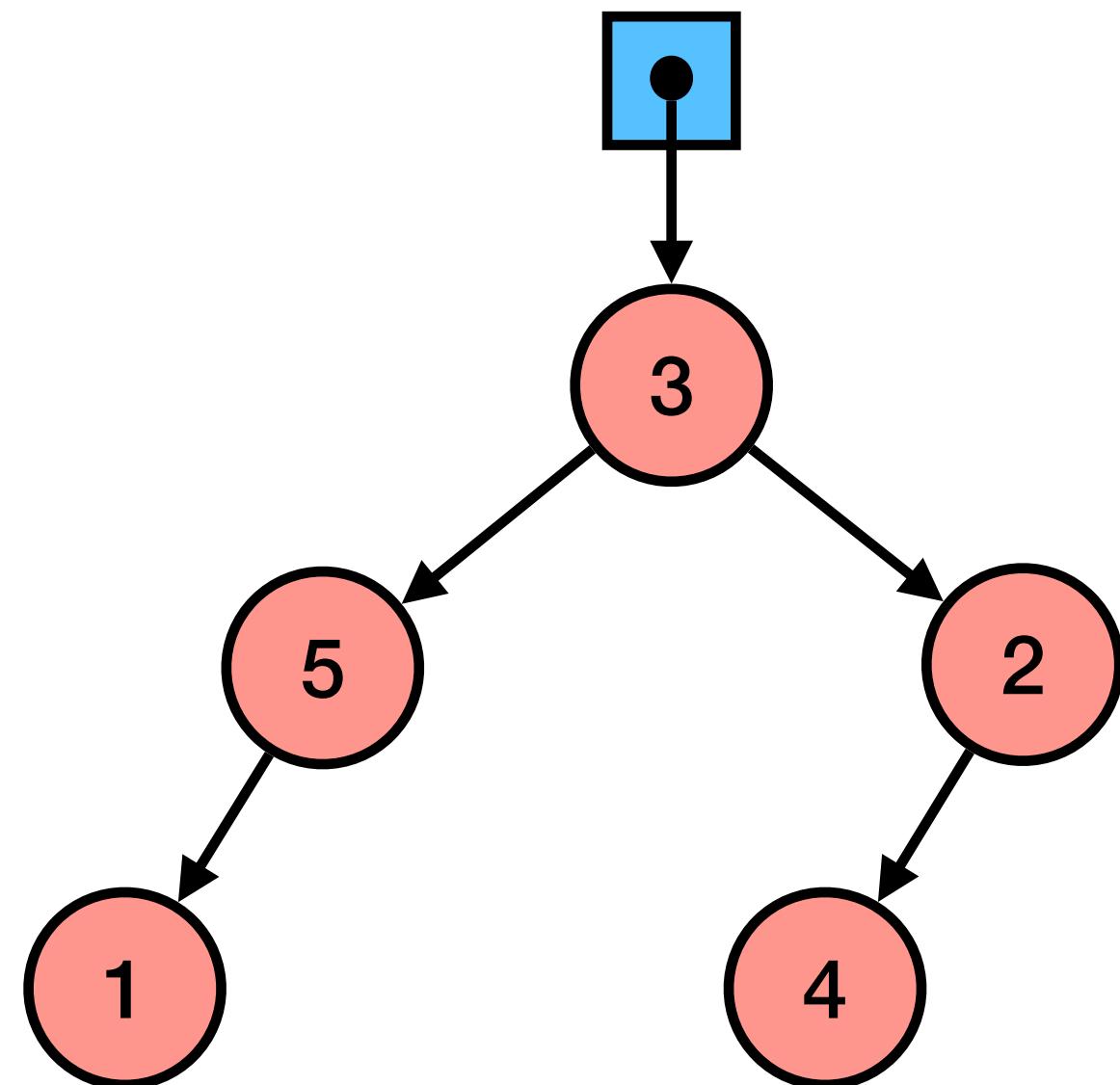
```
abin fromArray(int v[], int N) {  
    abin a;  
    if (N == 0) a = NULL;  
    else {  
        int m = N / 2;  
        abin l = fromArray(v, m);  
        abin r = fromArray(v+m+1, N-m-1);  
        a = mkroot(v[m], l, r);  
    }  
    return a;  
}
```

```
int main() {  
    int v[] = {5,3,8};  
    abin a = fromArray(v, 3);  
    return 0;  
}
```

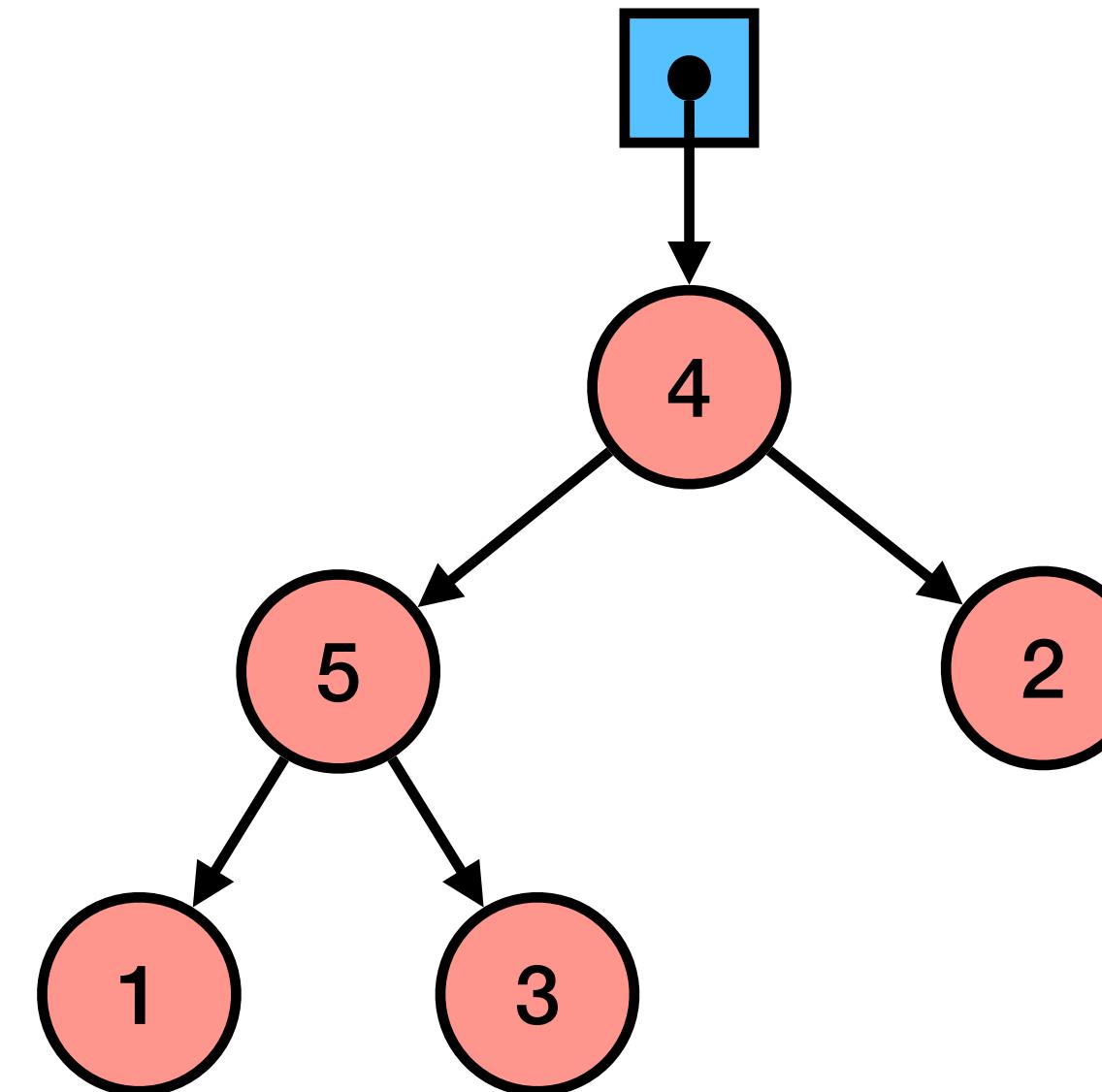


#27 Que árvore balanceada vai ser criada?

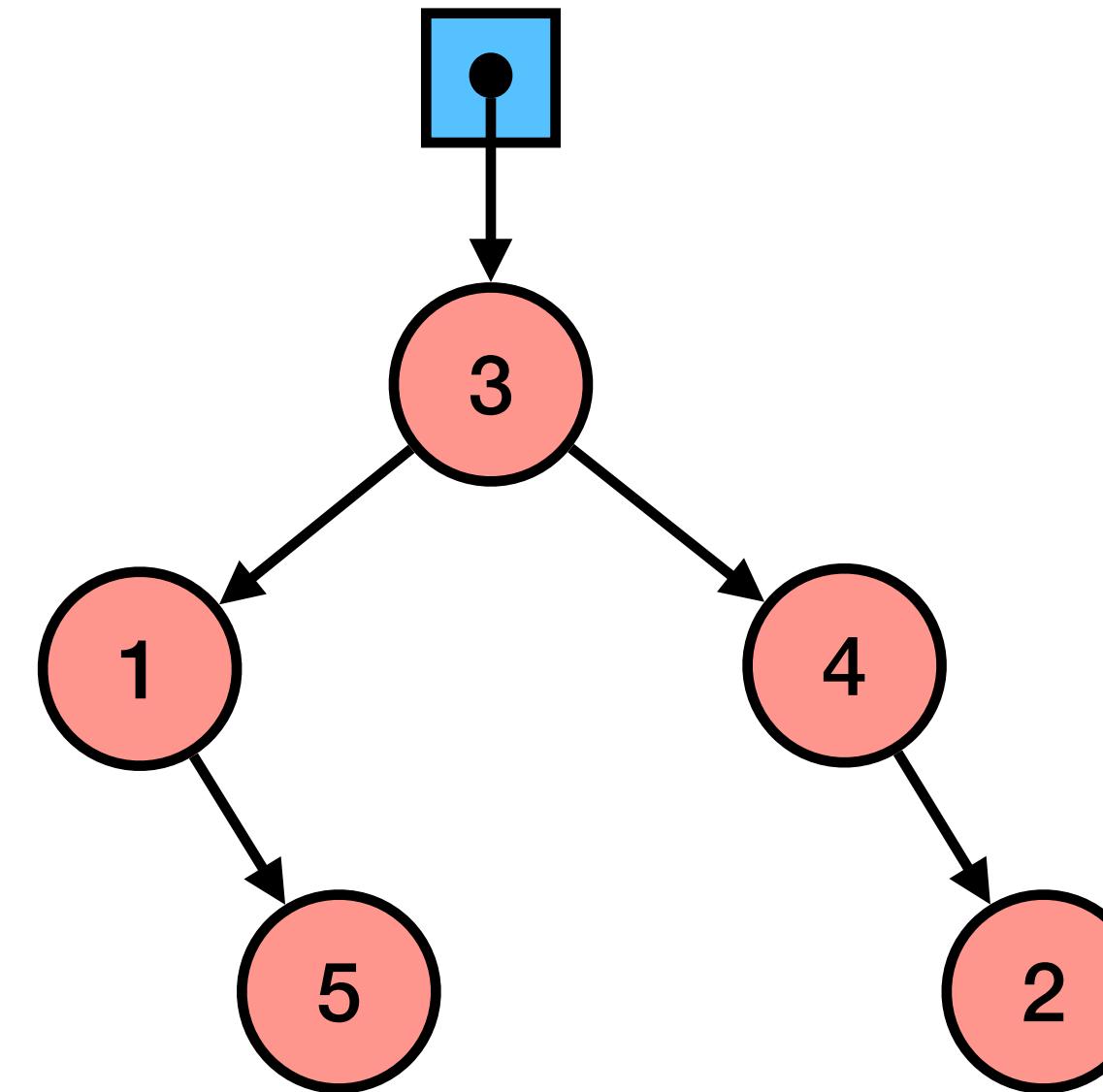
```
int v[] = {1,5,3,4,2}; abin a = fromArray(v, 5);
```



A



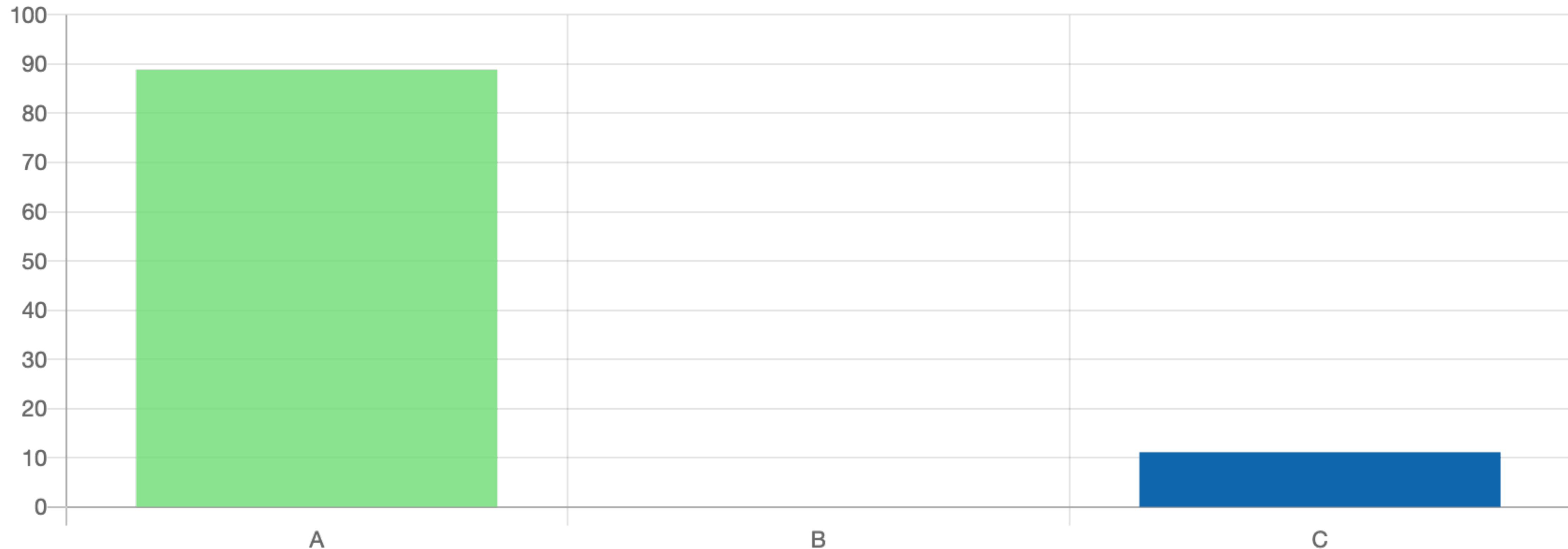
B



C

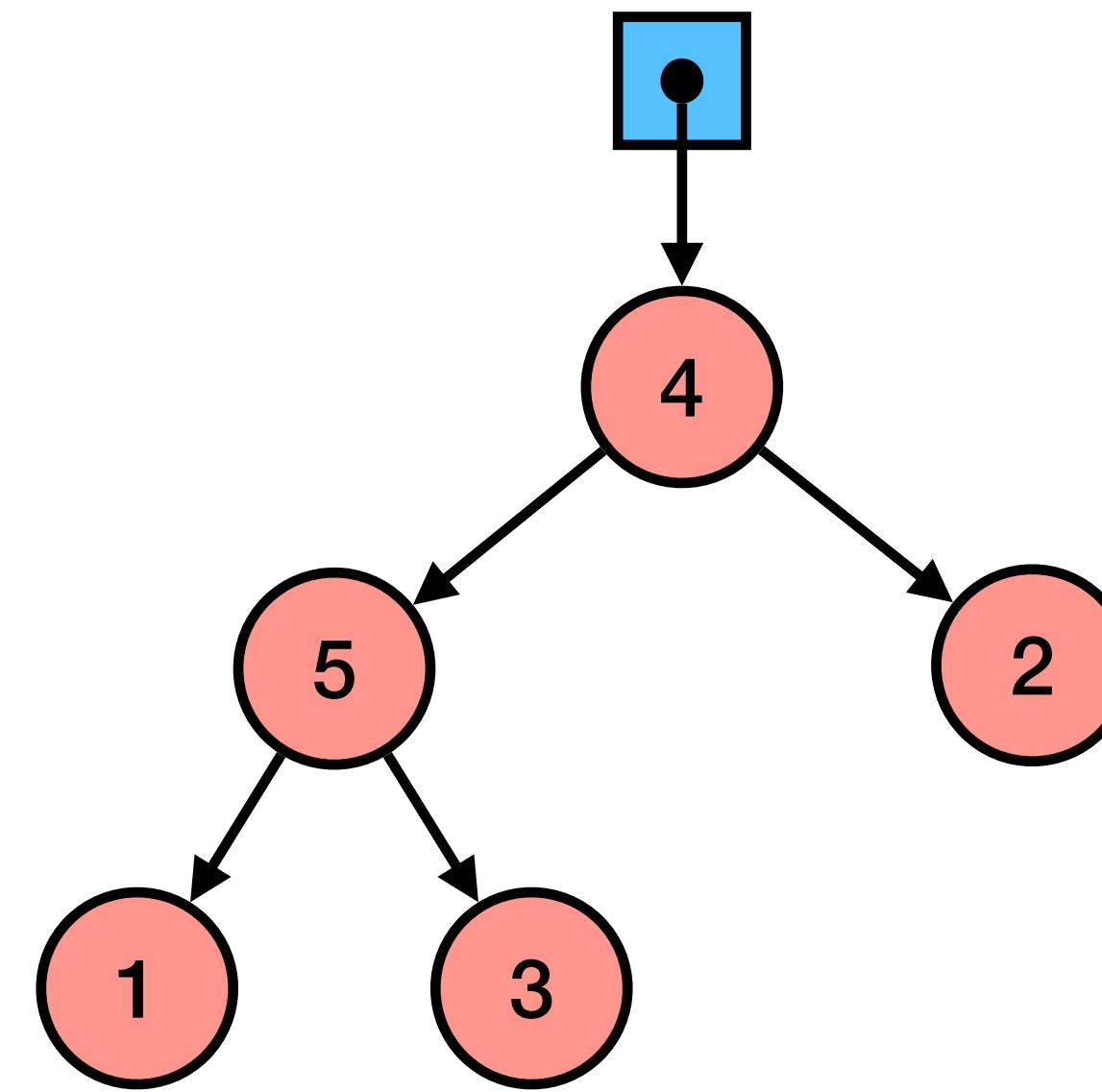


#27 Que árvore balanceada vai ser criada?



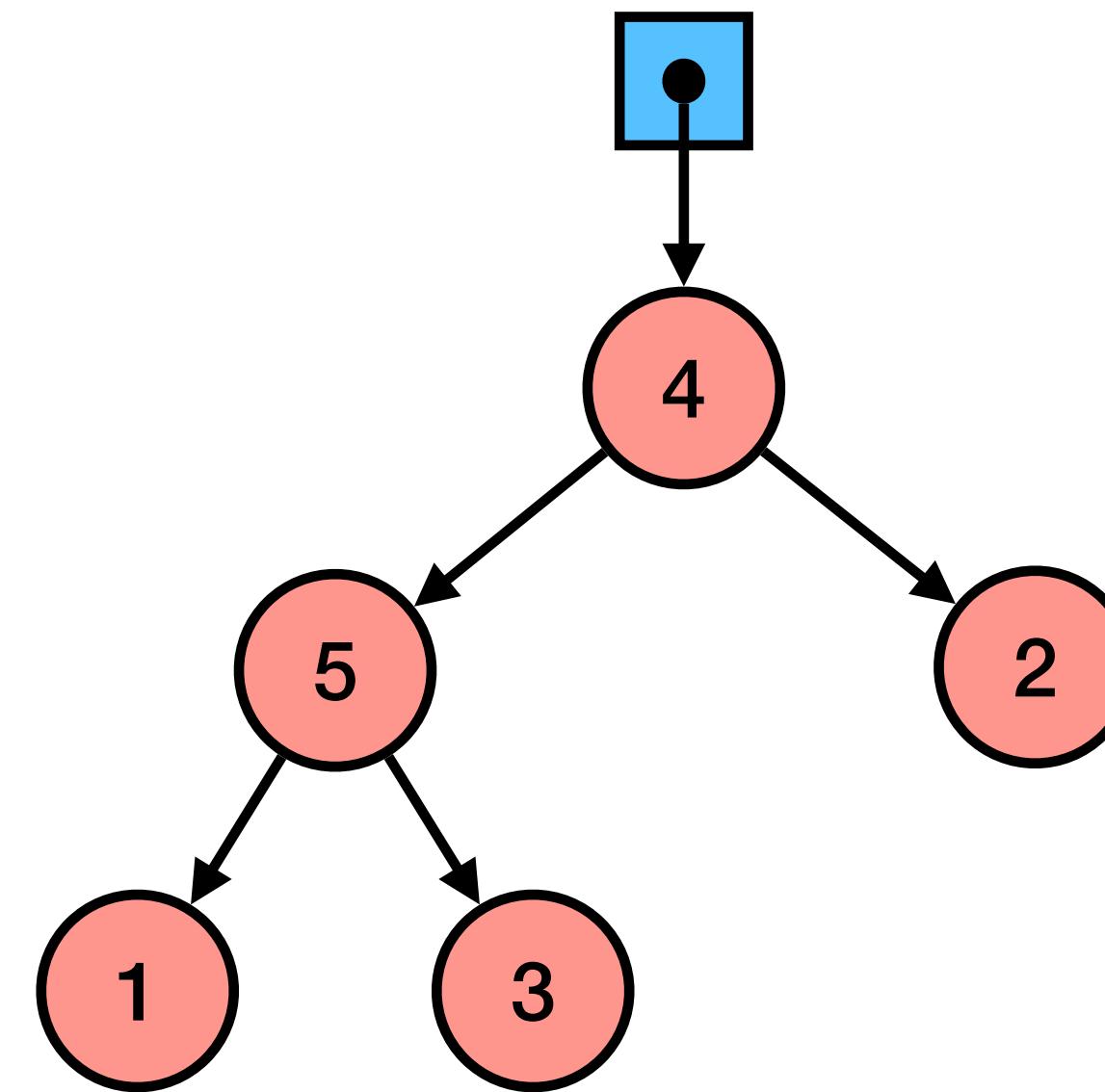
Travessia inorder

```
void inorder(abin a) {  
    if (a == NULL) return;  
    inorder(a->esq);  
    printf("%d\n", a->valor);  
    inorder(a->dir);  
}
```



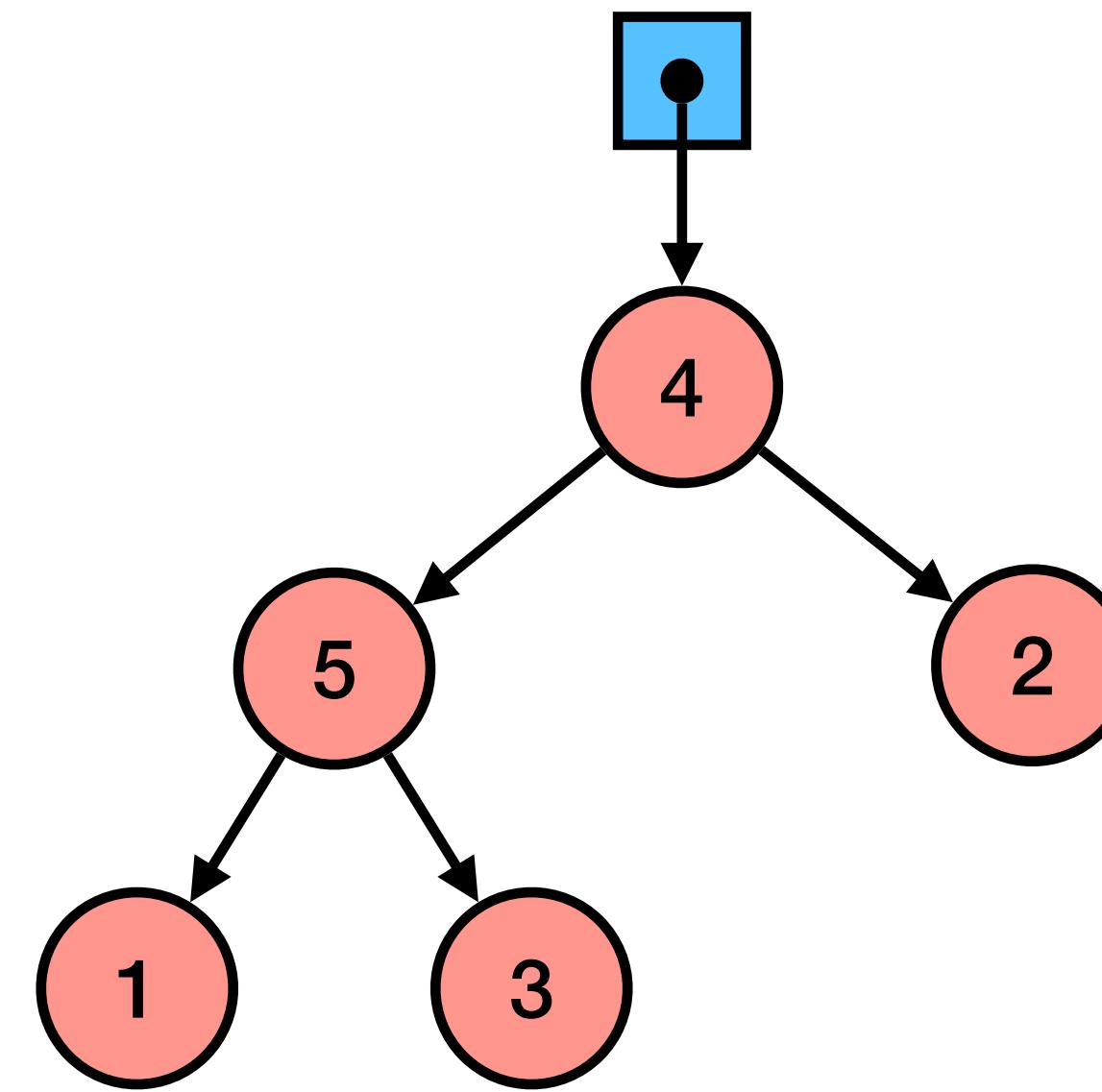
Travessia inorder

```
void inorder(abin a) {  
    if (a == NULL) return;  
    inorder(a->esq);  
    printf("%d\n", a->valor);  
    inorder(a->dir);  
}
```



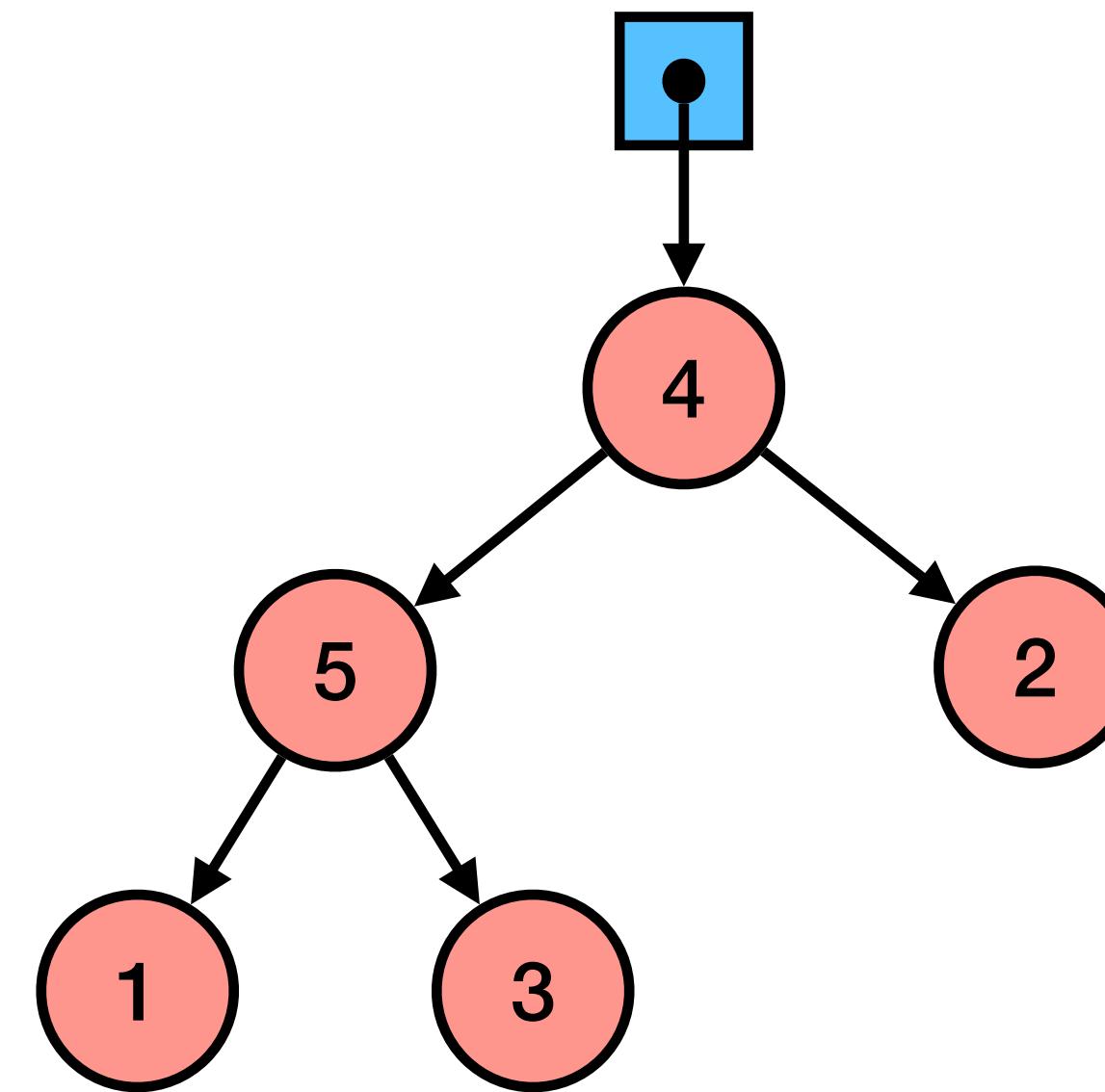
Travessia inorder

```
void inorder(abin a) {  
    if (a == NULL) return;  
    inorder(a->esq);  
    printf("%d\n", a->valor);  
    inorder(a->dir);  
}
```



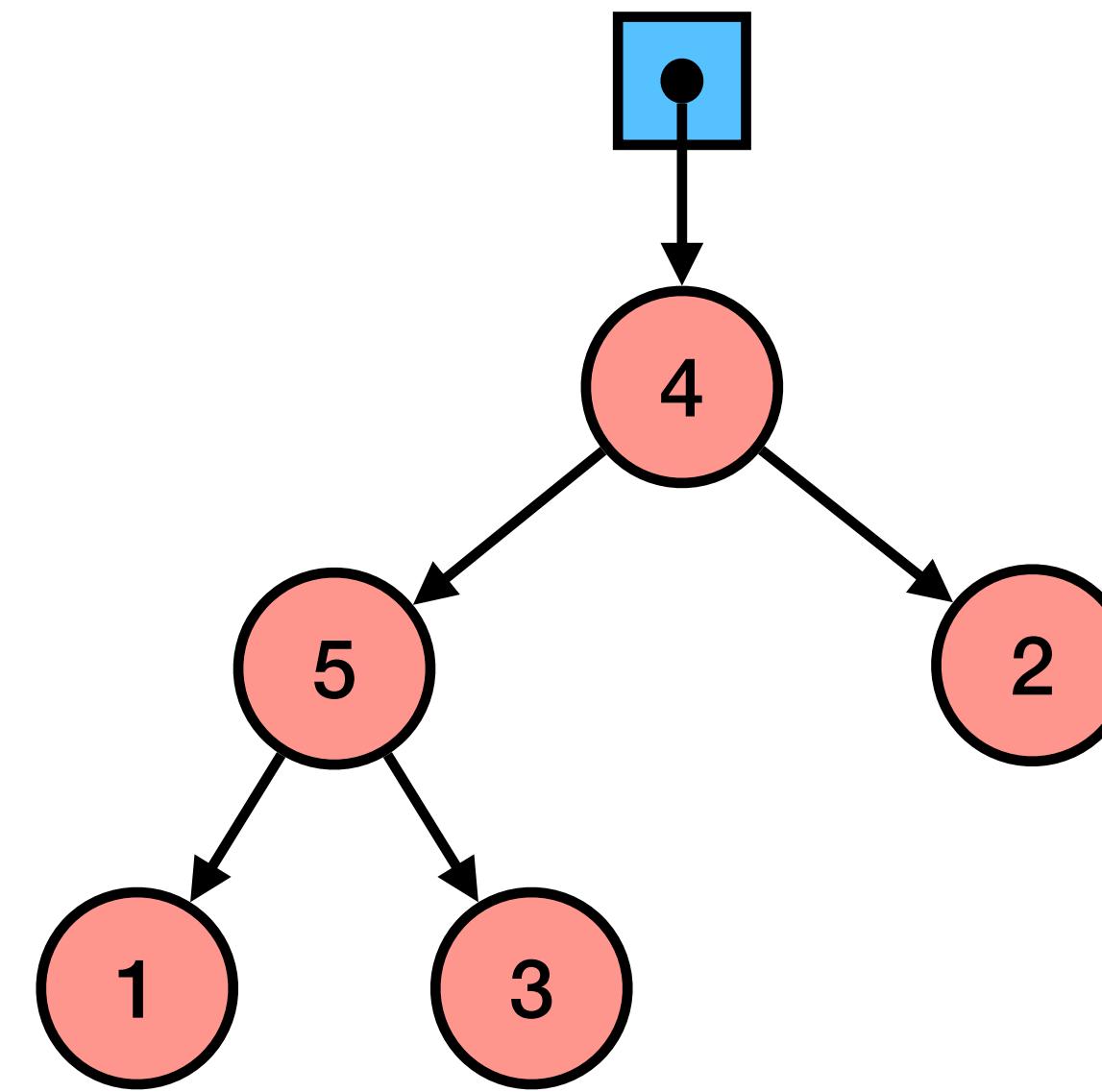
Travessia inorder

```
void inorder(abin a) {  
    if (a == NULL) return;  
    inorder(a->esq);  
    printf("%d\n", a->valor);  
    inorder(a->dir);  
}
```



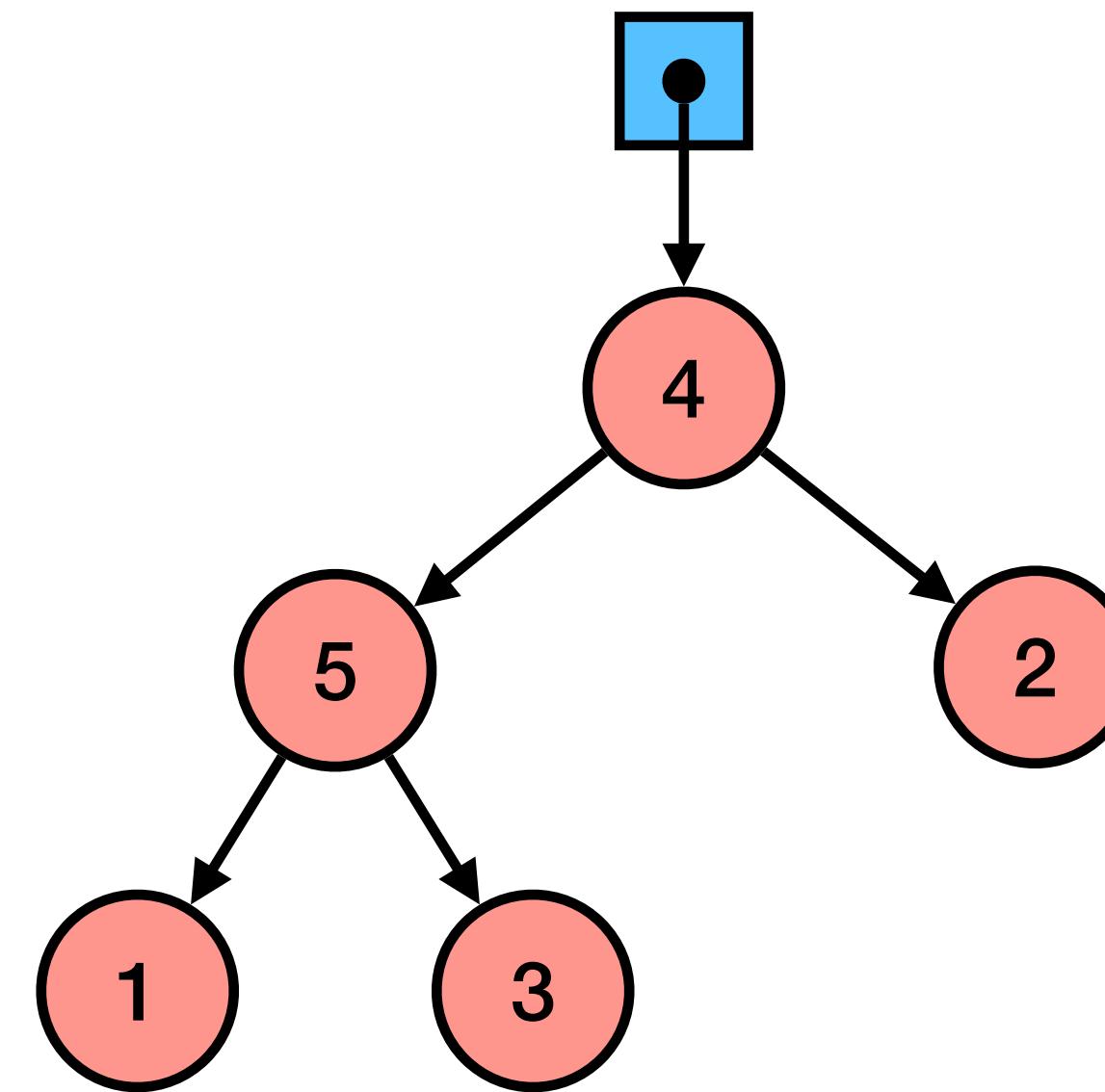
Travessia inorder

```
void inorder(abin a) {  
    if (a == NULL) return;  
    inorder(a->esq);  
    printf("%d\n", a->valor);  
    inorder(a->dir);  
}
```



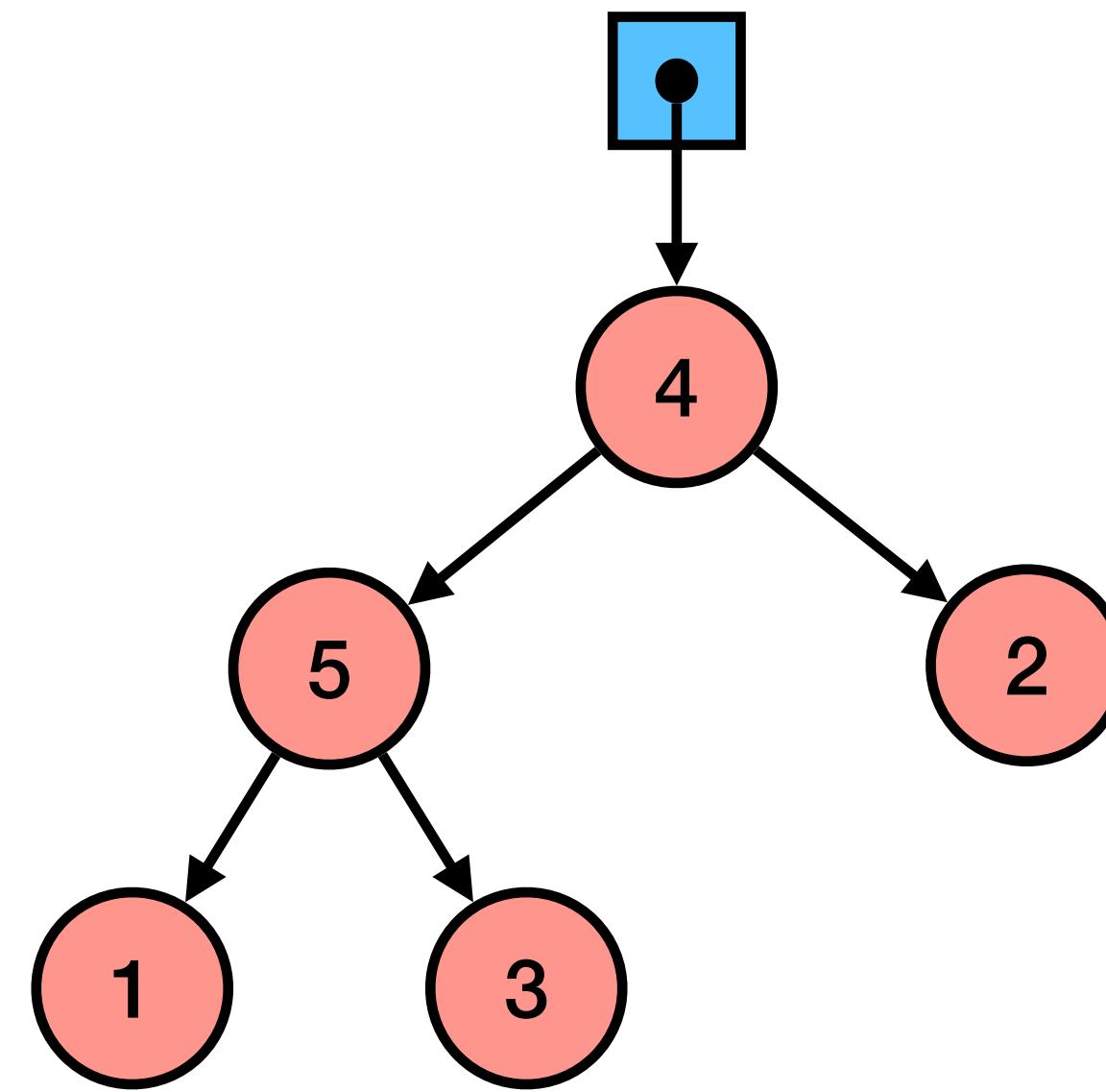
Travessia inorder

```
void inorder(abin a) {  
    if (a == NULL) return;  
    inorder(a->esq);  
    printf("%d\n", a->valor);  
    inorder(a->dir);  
}
```



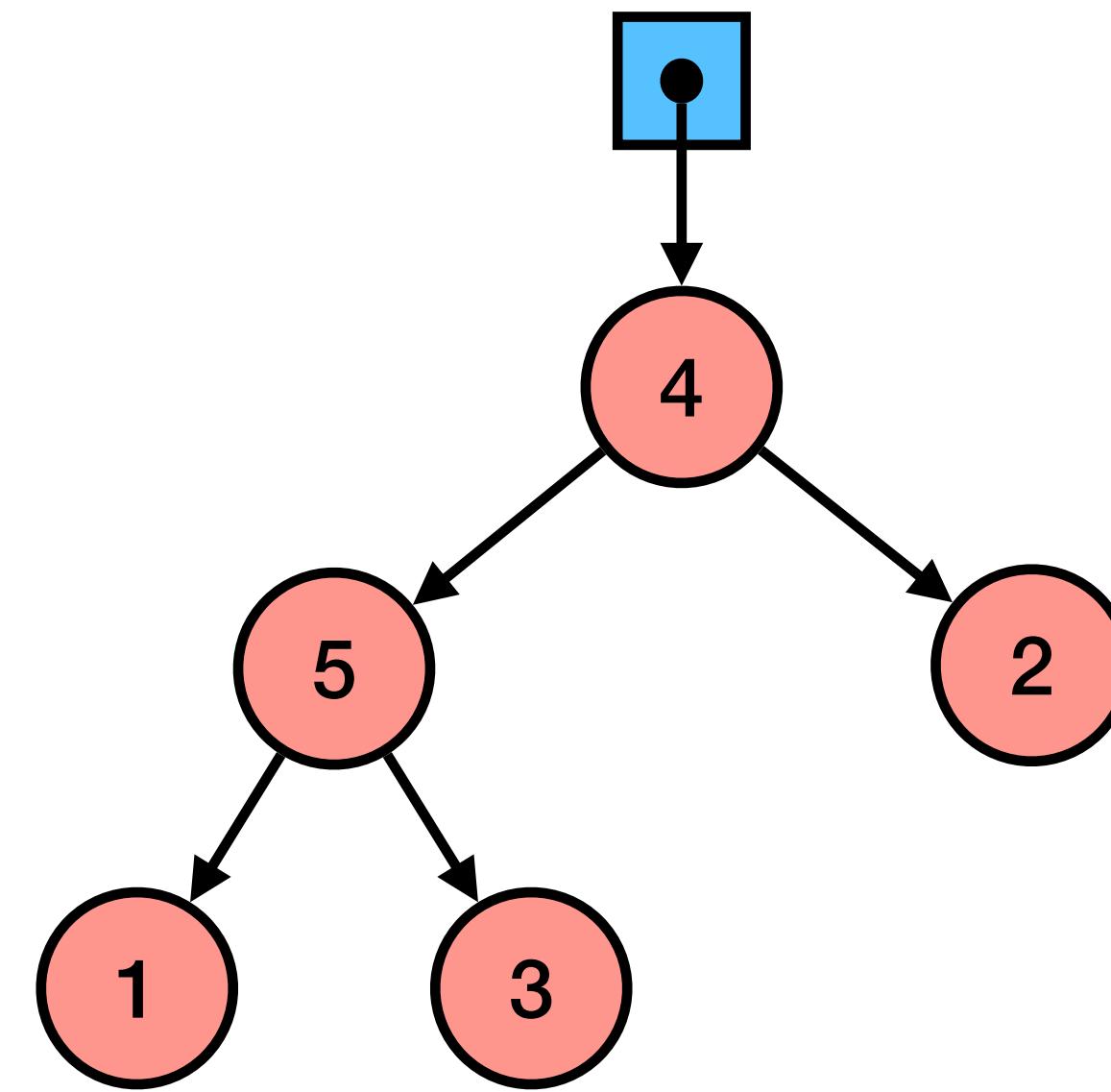
Travessia preorder

```
void preorder(abin a) {  
    if (a == NULL) return;  
    printf("%d\n", a->valor);  
    preorder(a->esq);  
    preorder(a->dir);  
}
```



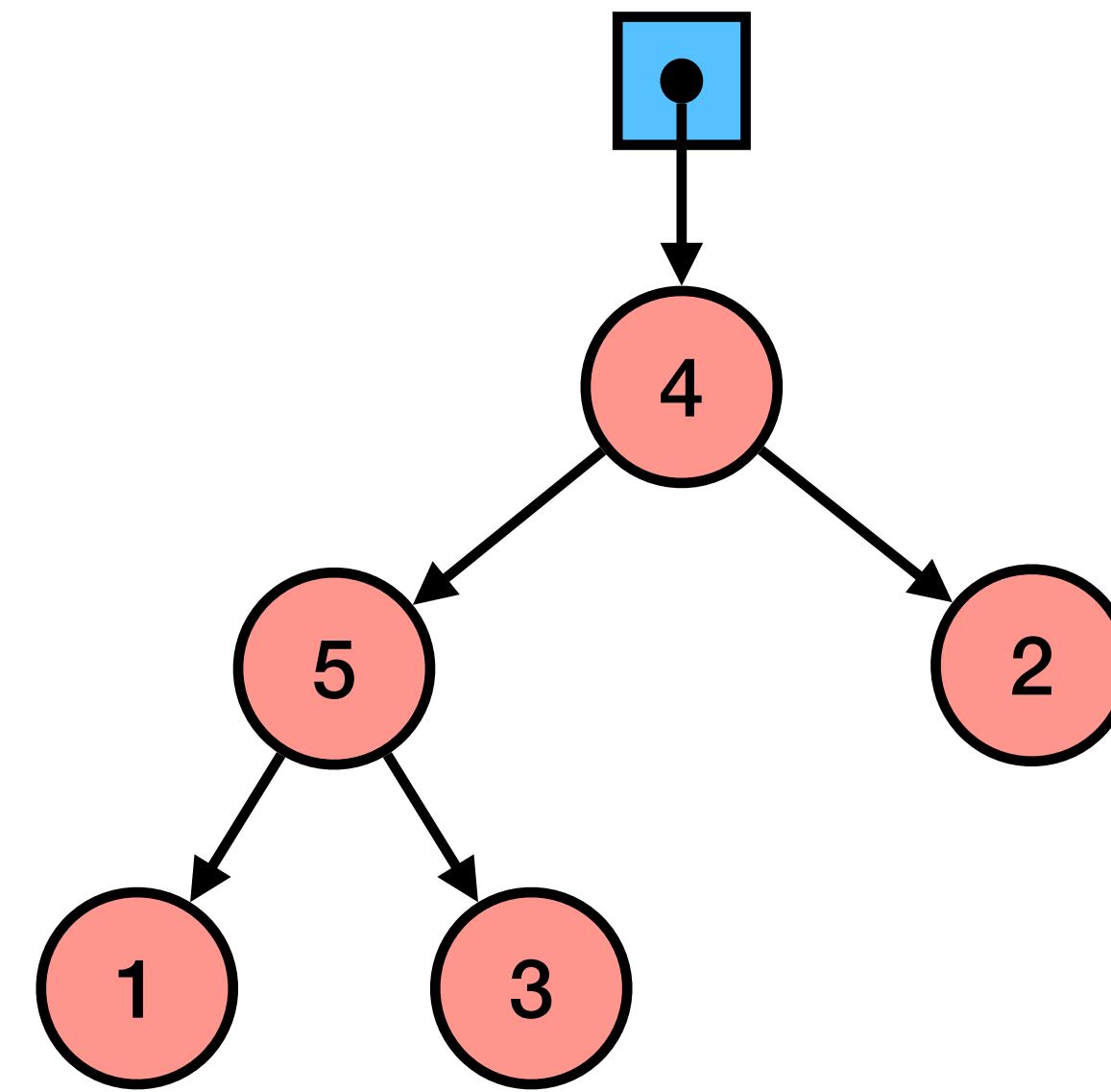
Travessia preorder

```
void preorder(abin a) {  
    if (a == NULL) return;  
    printf("%d\n", a->valor);  
    preorder(a->esq);  
    preorder(a->dir);  
}
```



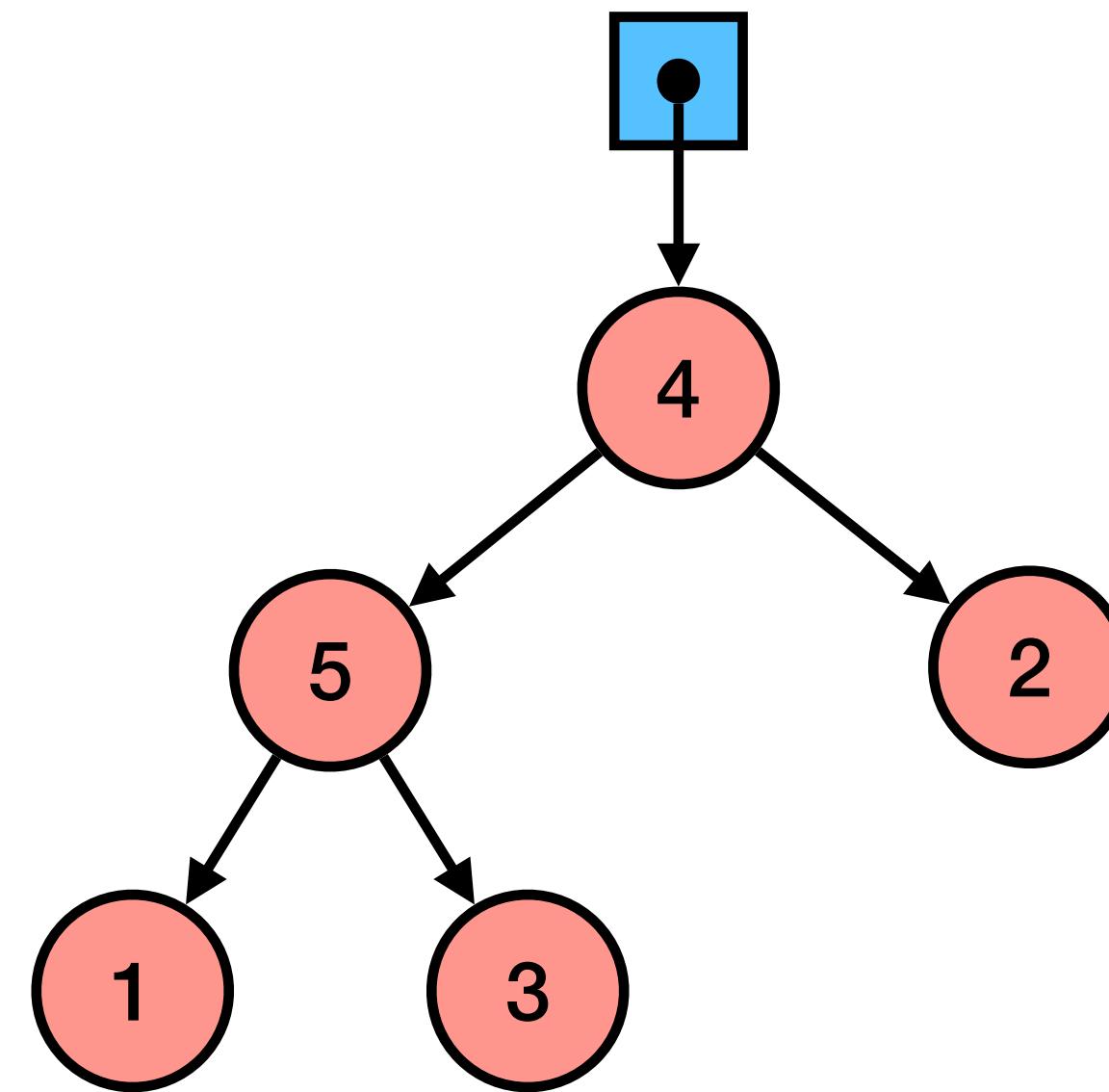
Travessia preorder

```
void preorder(abin a) {  
    if (a == NULL) return;  
    printf("%d\n", a->valor);  
    preorder(a->esq);  
    preorder(a->dir);  
}
```



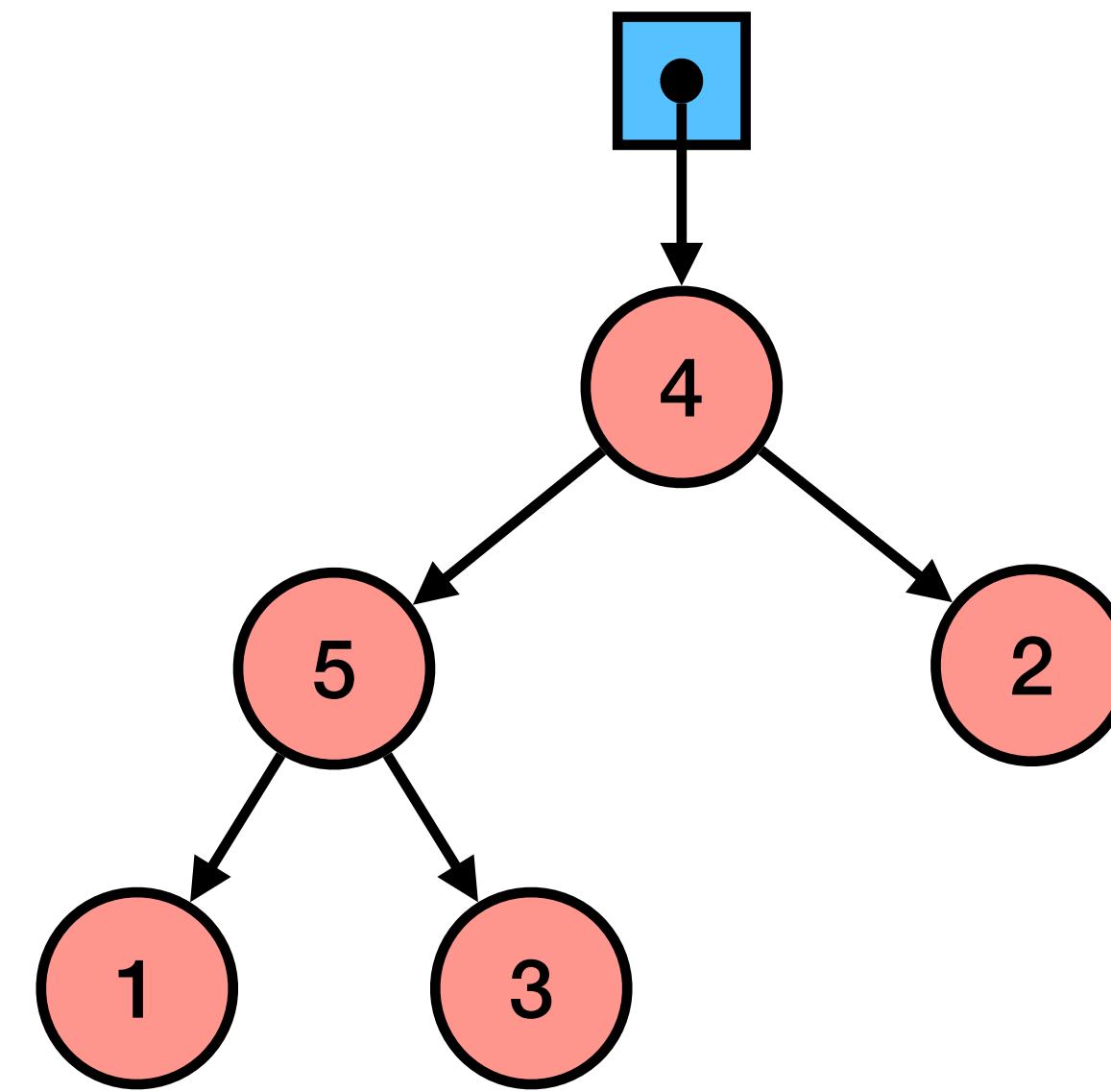
Travessia preorder

```
void preorder(abin a) {  
    if (a == NULL) return;  
    printf("%d\n", a->valor);  
    preorder(a->esq);  
    preorder(a->dir);  
}
```



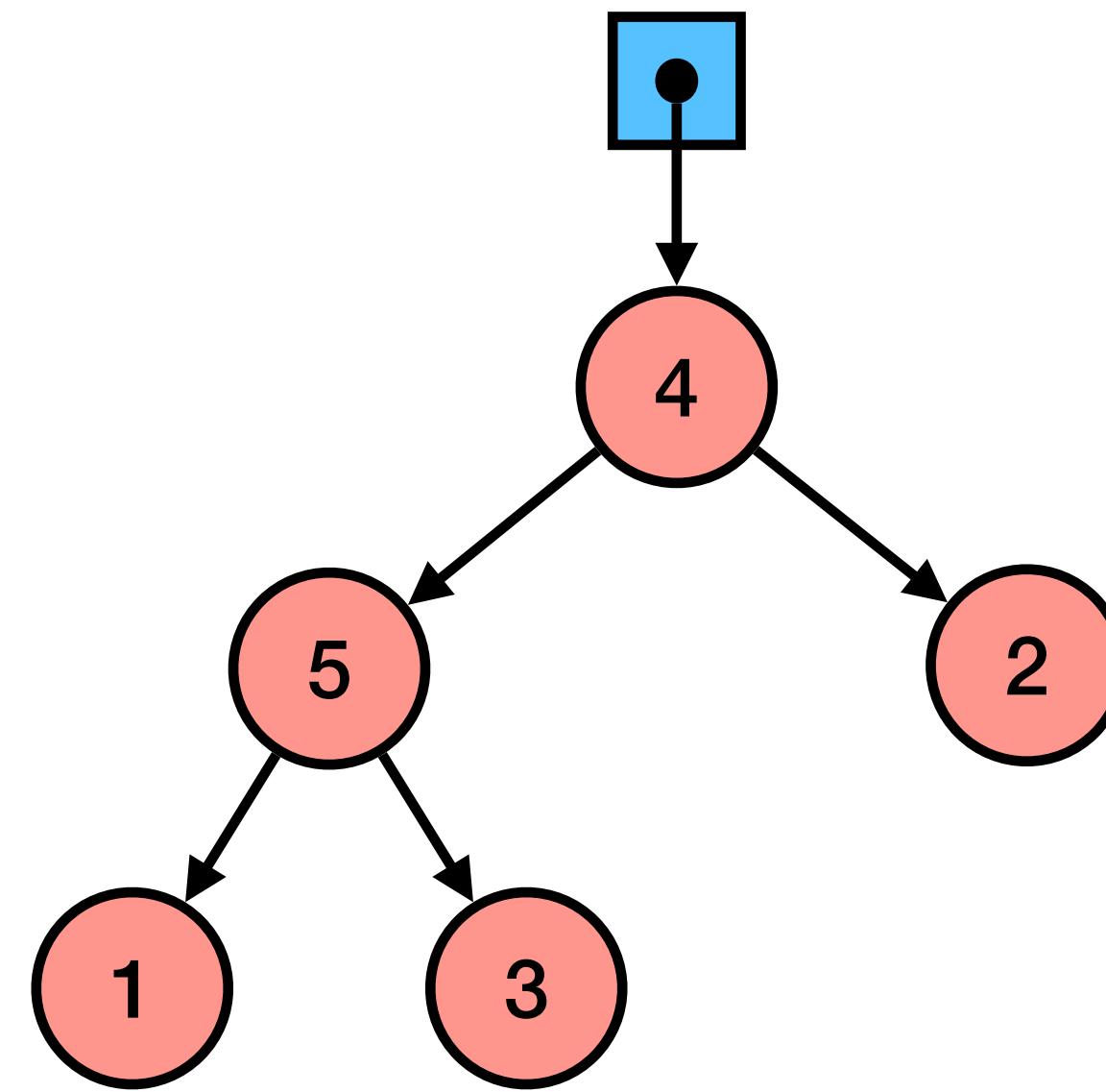
Travessia preorder

```
void preorder(abin a) {  
    if (a == NULL) return;  
    printf("%d\n", a->valor);  
    preorder(a->esq);  
    preorder(a->dir);  
}
```



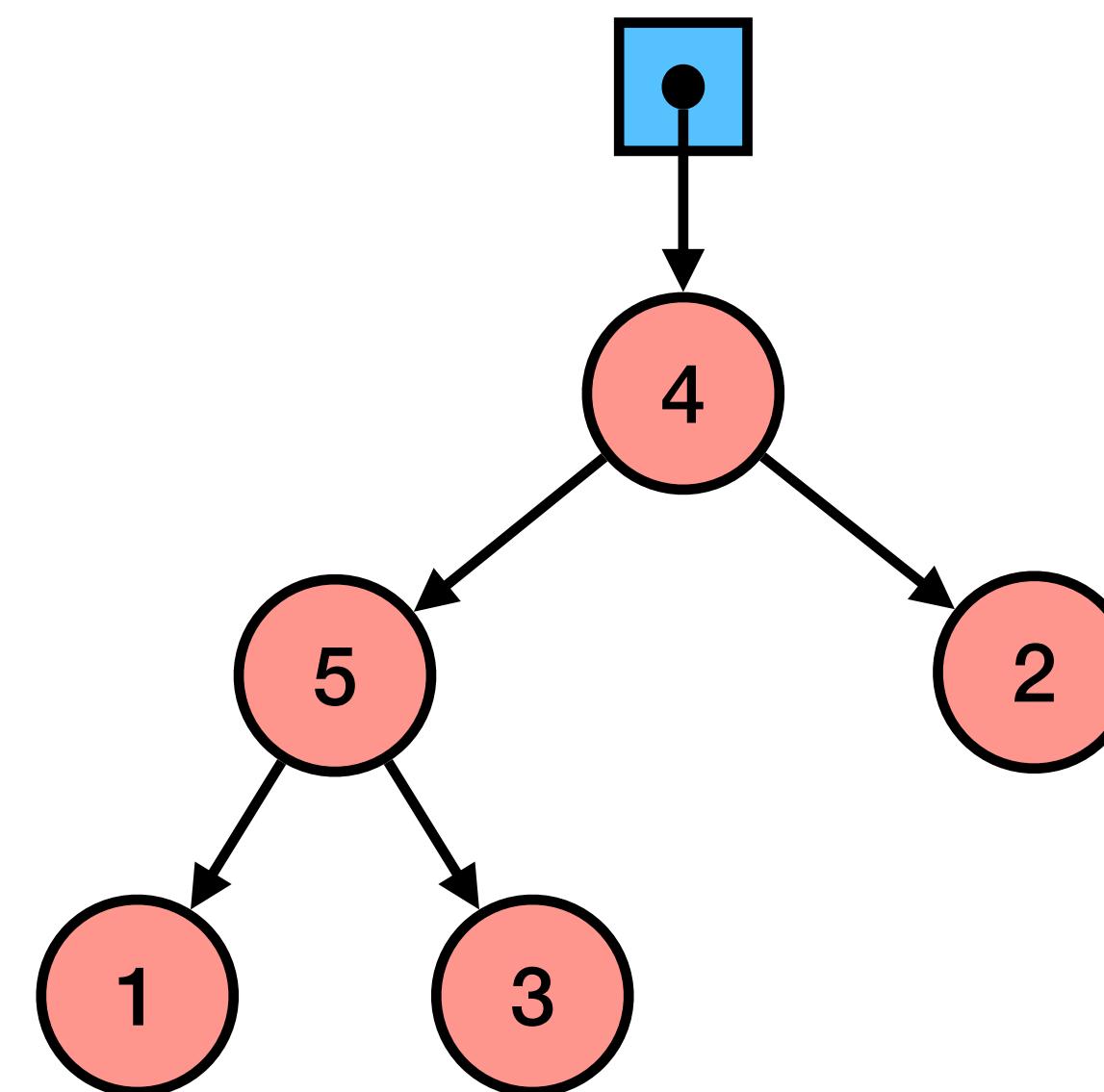
Travessia preorder

```
void preorder(abin a) {  
    if (a == NULL) return;  
    printf("%d\n", a->valor);  
    preorder(a->esq);  
    preorder(a->dir);  
}
```



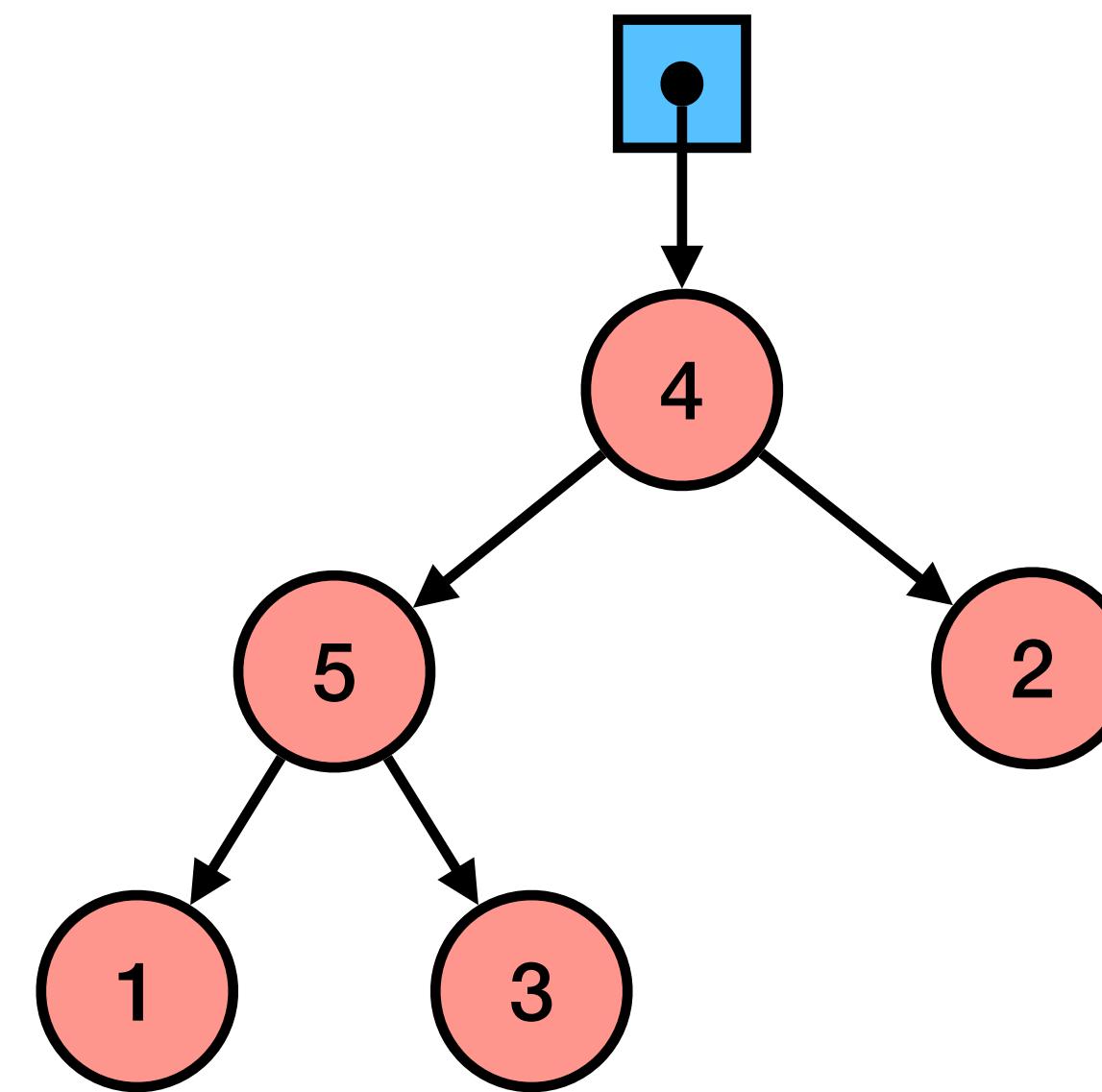
#28 Qual a 3^a linha impressa?

```
void postorder(abin a) {  
    if (a == NULL) return;  
    postorder(a->esq);  
    postorder(a->dir);  
    printf("%d\n", a->valor);  
}
```



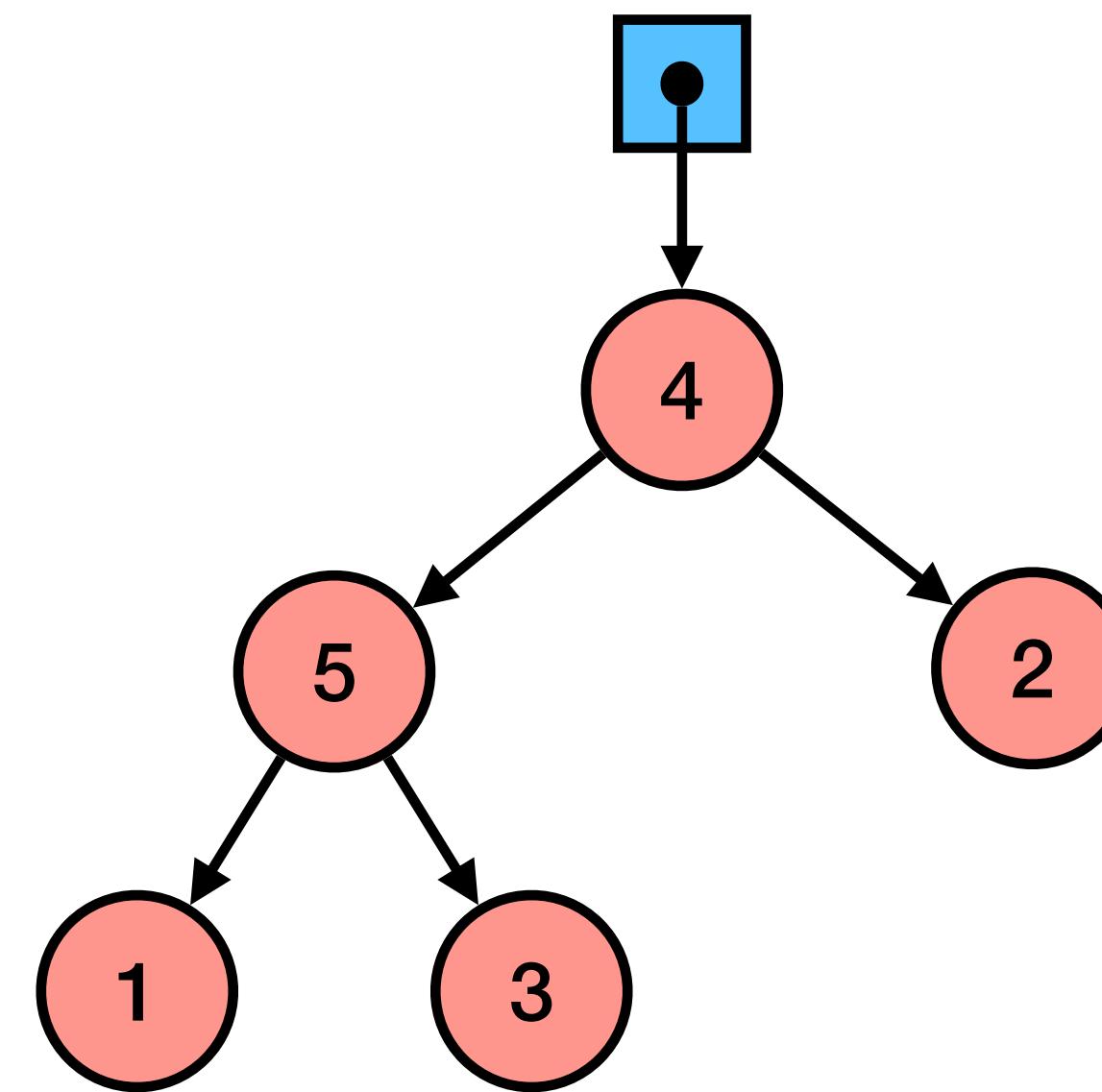
Travessia postorder

```
void postorder(abin a) {  
    if (a == NULL) return;  
    postorder(a->esq);  
    postorder(a->dir);  
    printf("%d\n", a->valor);  
}
```



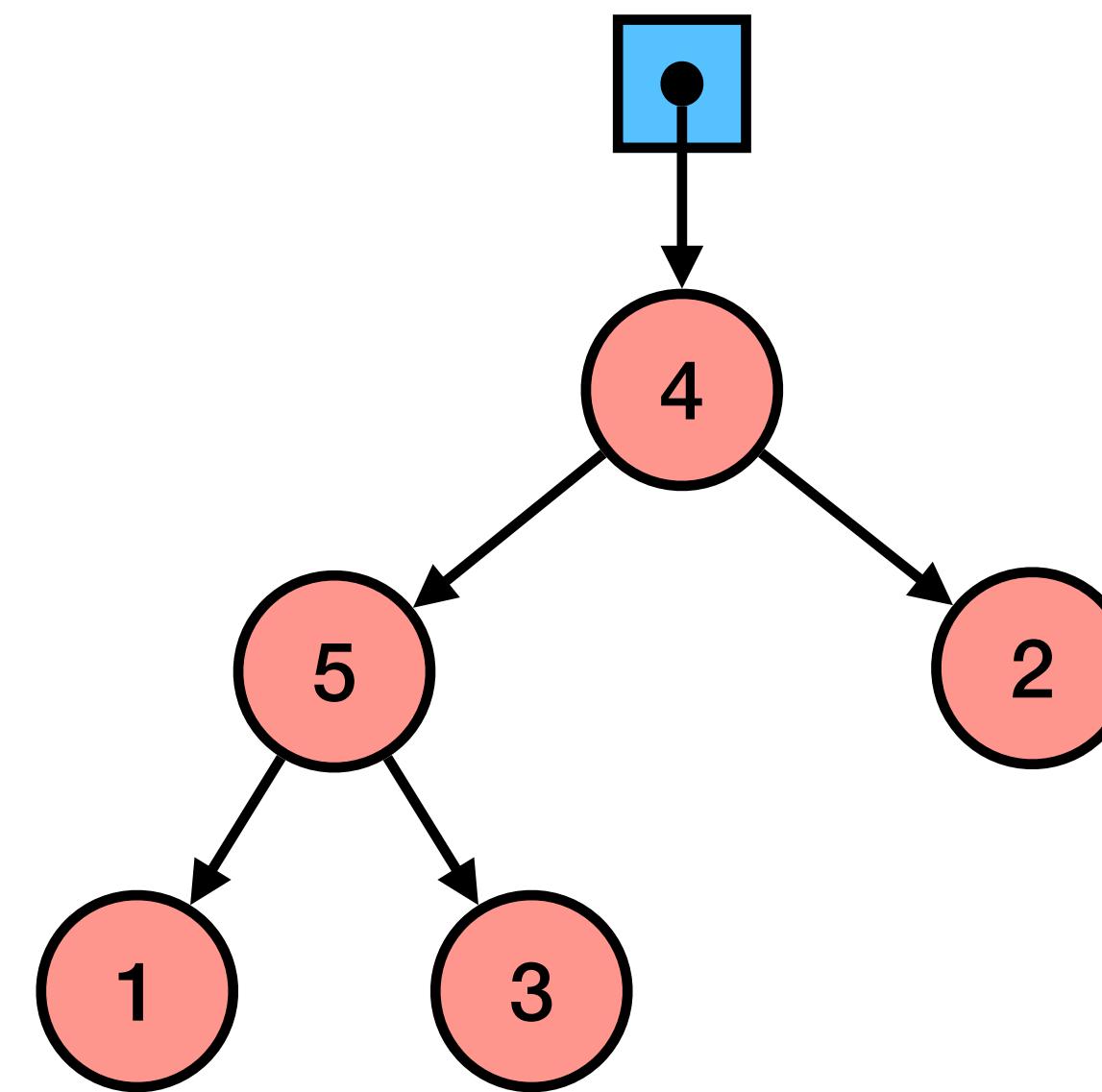
Travessia postorder

```
void postorder(abin a) {  
    if (a == NULL) return;  
    postorder(a->esq);  
    postorder(a->dir);  
    printf("%d\n", a->valor);  
}
```



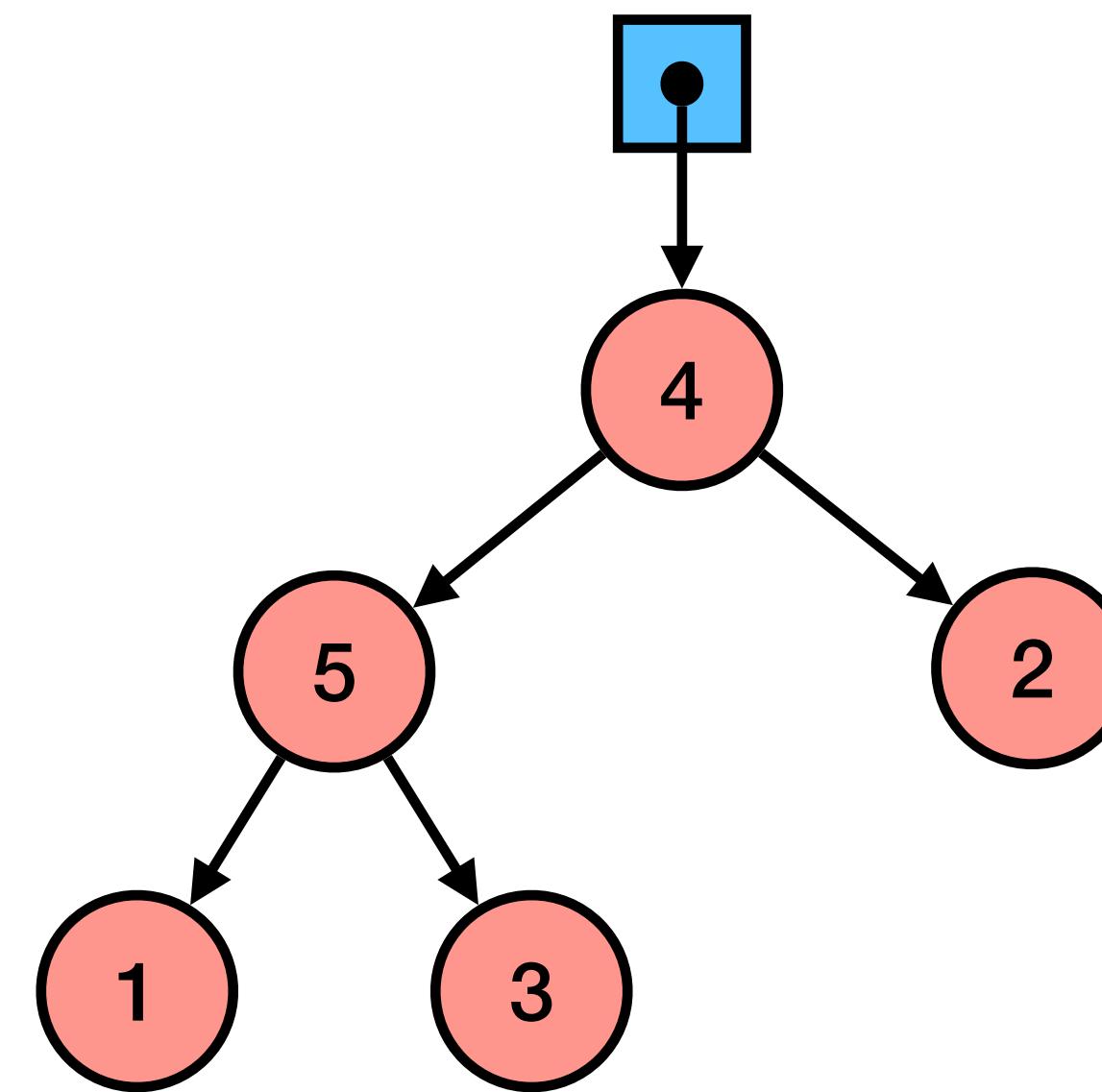
Travessia postorder

```
void postorder(abin a) {  
    if (a == NULL) return;  
    postorder(a->esq);  
    postorder(a->dir);  
    printf("%d\n", a->valor);  
}
```



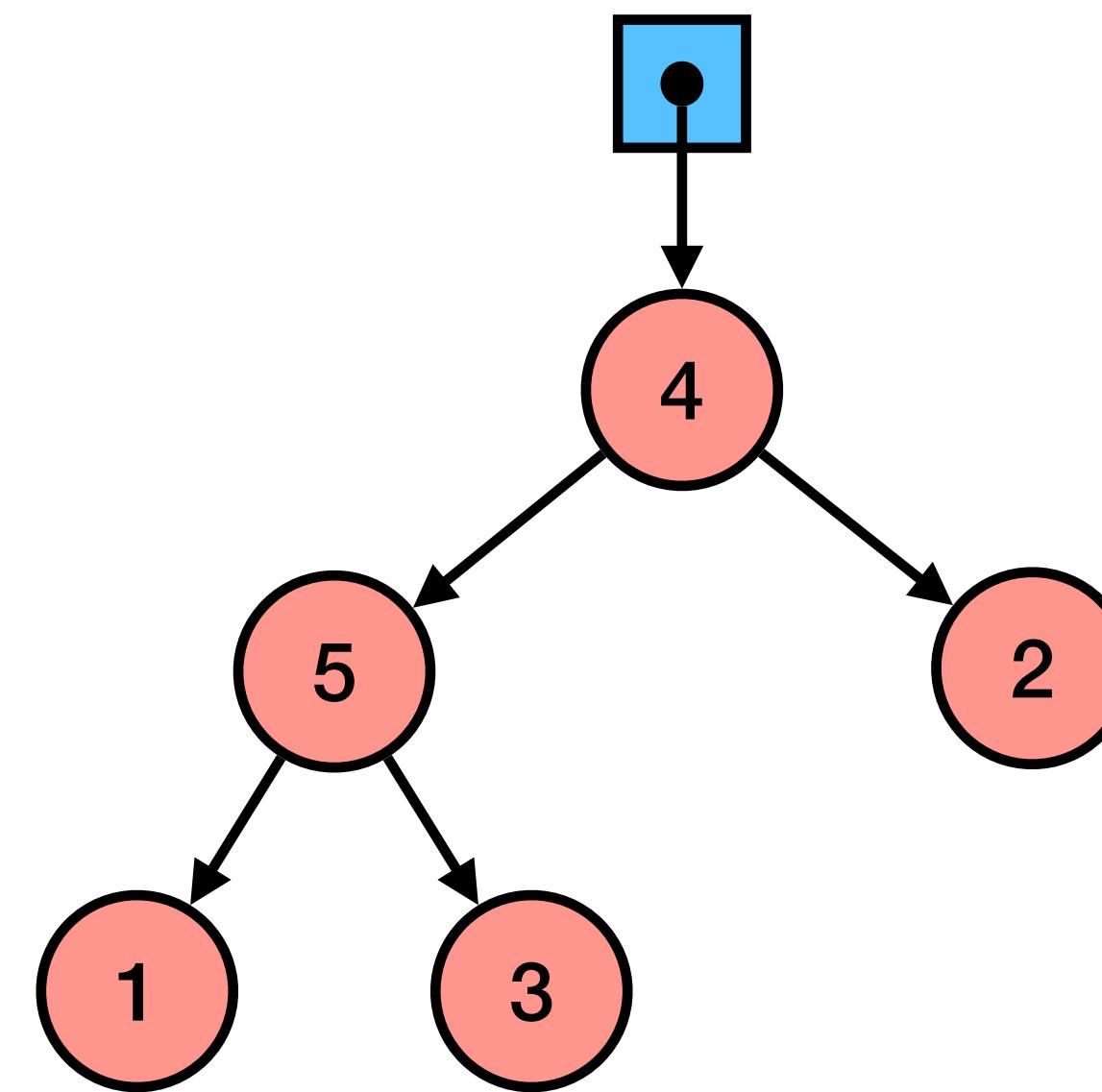
Travessia postorder

```
void postorder(abin a) {  
    if (a == NULL) return;  
    postorder(a->esq);  
    postorder(a->dir);  
    printf("%d\n", a->valor);  
}
```



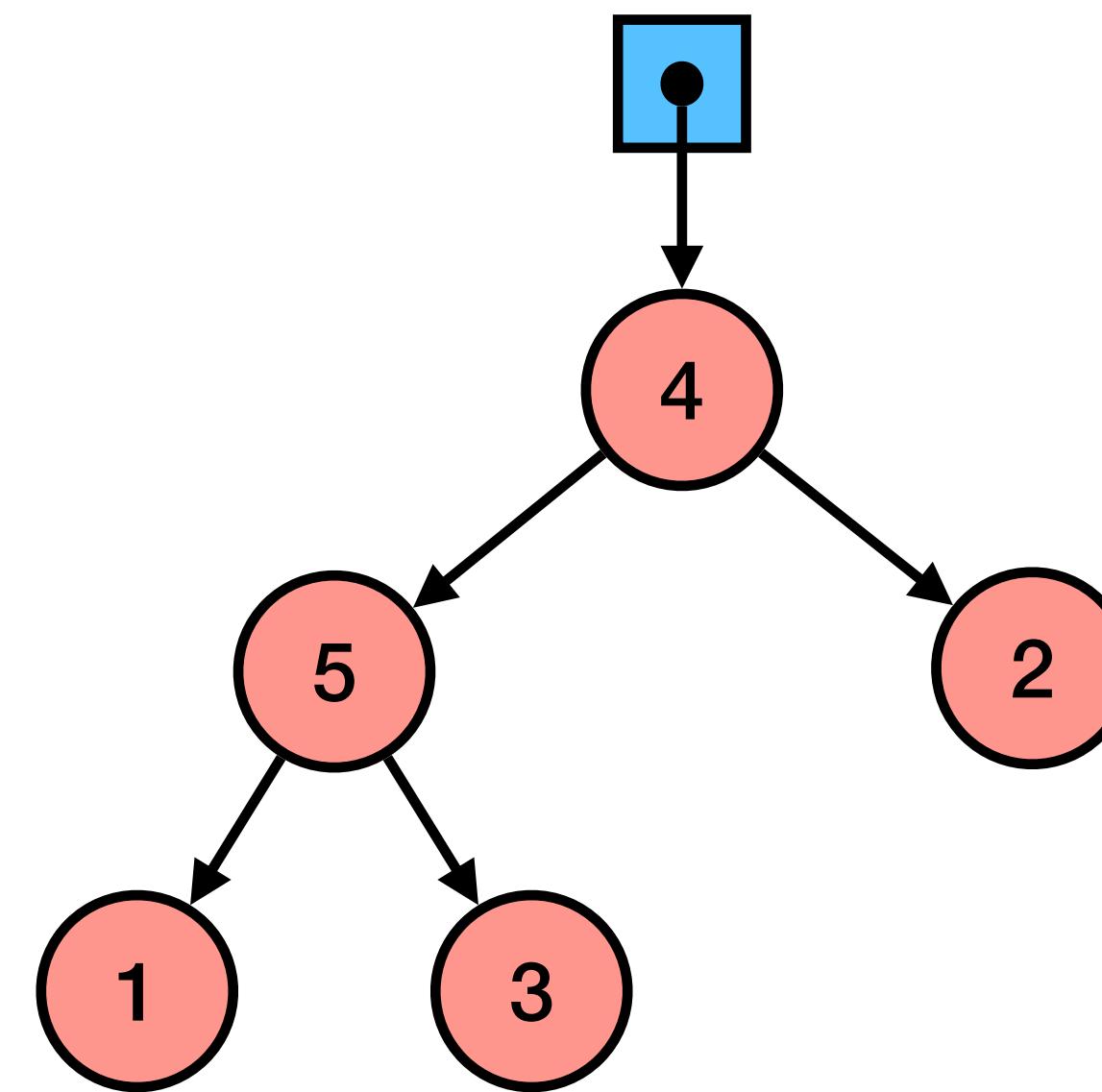
Travessia postorder

```
void postorder(abin a) {  
    if (a == NULL) return;  
    postorder(a->esq);  
    postorder(a->dir);  
    printf("%d\n", a->valor);  
}
```



Travessia postorder

```
void postorder(abin a) {  
    if (a == NULL) return;  
    postorder(a->esq);  
    postorder(a->dir);  
    printf("%d\n", a->valor);  
}
```



Copiar para array inorder

```
int toArrayInorder(abin a, int v[]) {  
    int l, r;  
    if (a == NULL) return 0;  
    l = toArrayInorder(a->esq, v);  
    v[l] = a->valor;  
    r = toArrayInorder(a->dir, v+1+l);  
    return 1+l+r;  
}
```

Copiar para array preorder

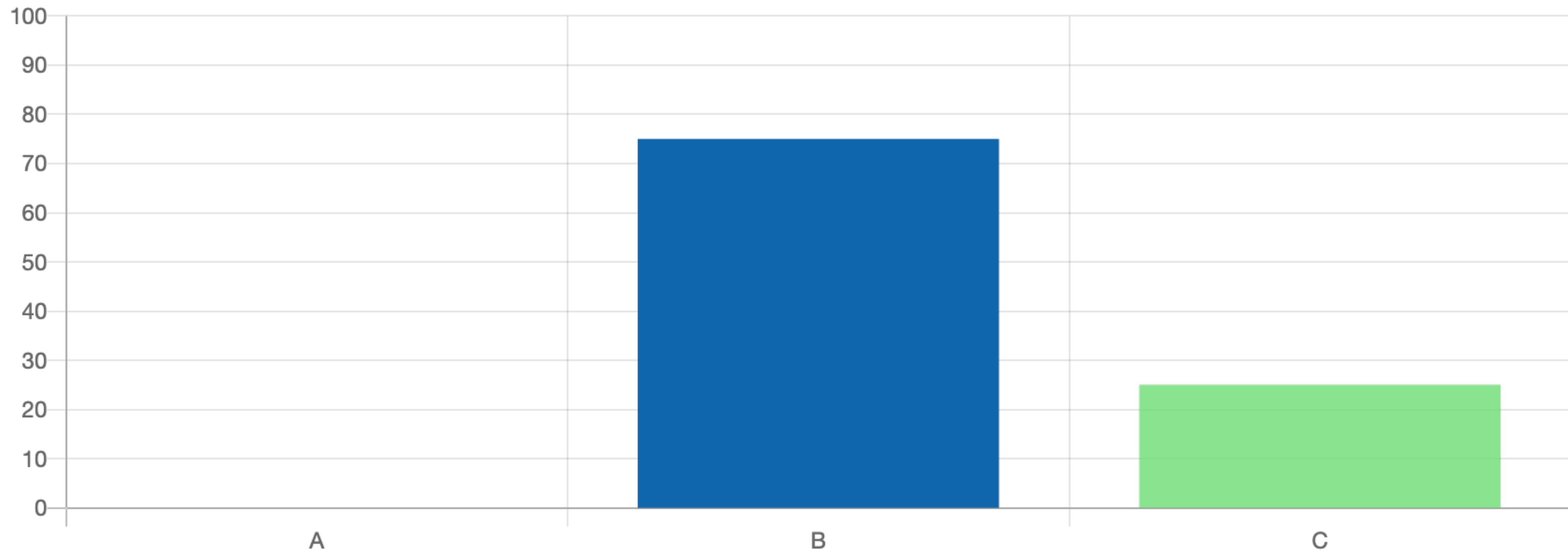
```
int toArrayPreorder(abin a, int v[]) {  
    int l, r;  
    if (a == NULL) return 0;  
    v[0] = a->valor;  
    l = toArrayPreorder(a->esq, v+1);  
    r = toArrayPreorder(a->dir, v+1+l);  
    return 1+l+r;  
}
```

#29 Como copiar para array postorder?

```
int A(abin a, int v[]) {      int B(abin a, int v[]) {      int C(abin a, int v[]) {
    int l, r;                  int l, r;                  int l, r;
    if (a == NULL) return 0;    if (a == NULL) return 0;    if (a == NULL) return 0;
    l = A(a->esq, v+1);     l = B(a->esq, v);     l = C(a->esq, v);
    r = A(a->dir, v+1+l);   r = B(a->dir, v+1);   r = C(a->dir, v+1);
    v[0] = a->valor;         v[1+l+r] = a->valor; v[1+r] = a->valor;
    return 1+l+r;              return 1+l+r;           return 1+l+r;
}                                }                            }
```



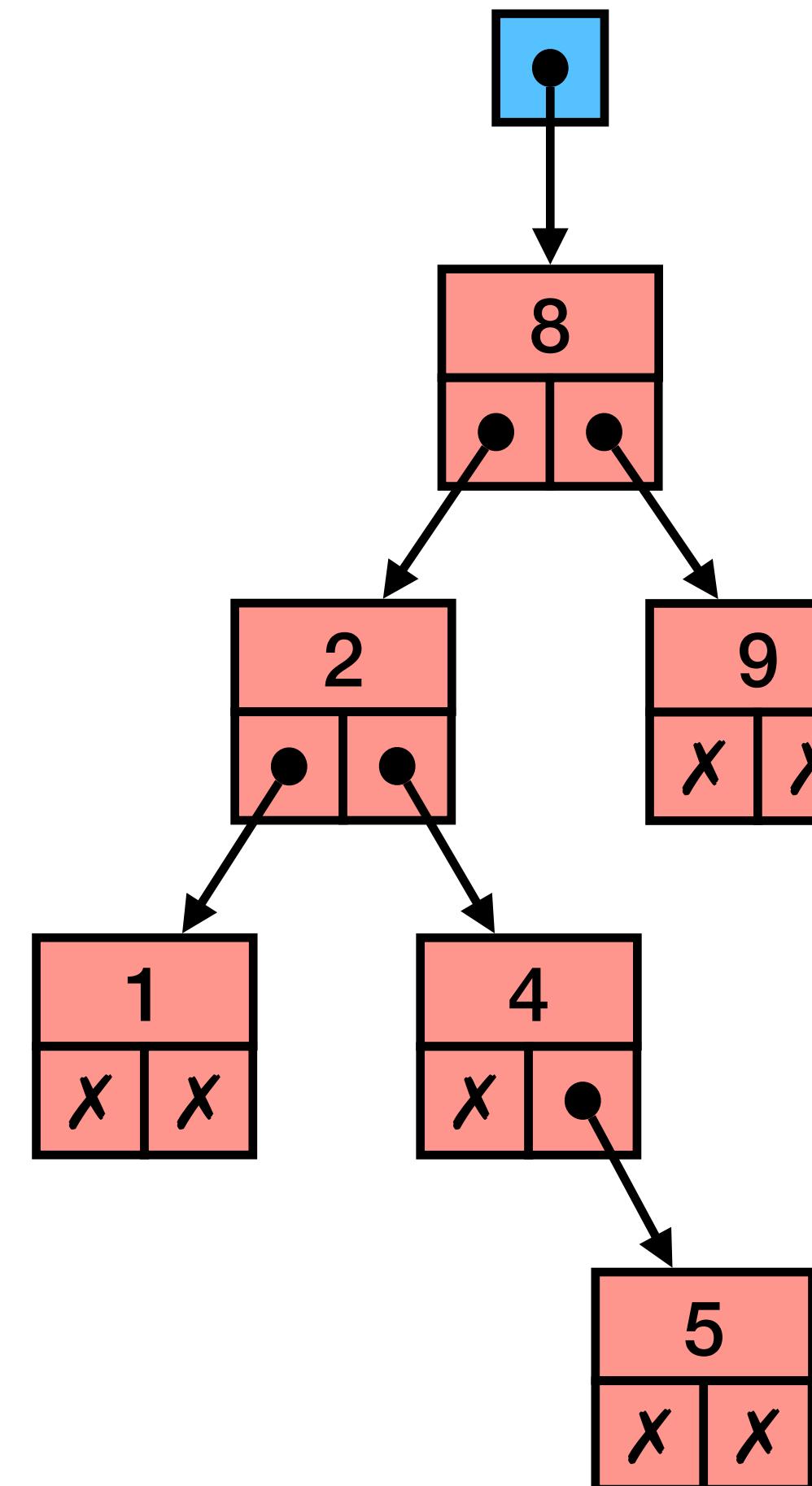
#29 Como copiar para array postorder?



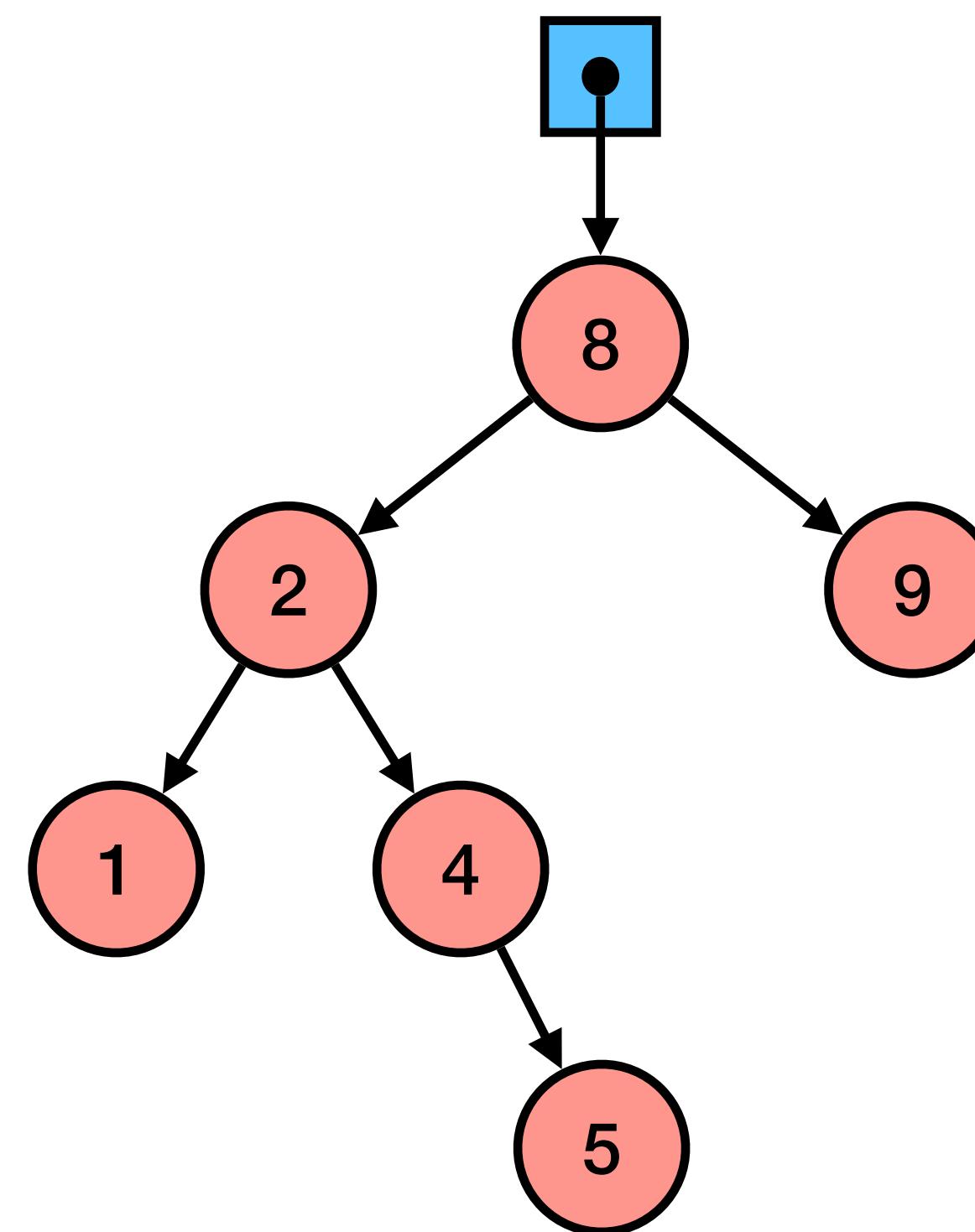
Pesquisa

```
abin search(int x, abin a) {  
    abin r;  
    if (a == NULL) return NULL;  
    if (a->valor == x) return a;  
    r = search(x, a->esq);  
    if (r == NULL) return search(x, a->dir);  
    return r;  
}
```

Árvores binárias de procura



Árvores binárias de procura



Pesquisa ordenada

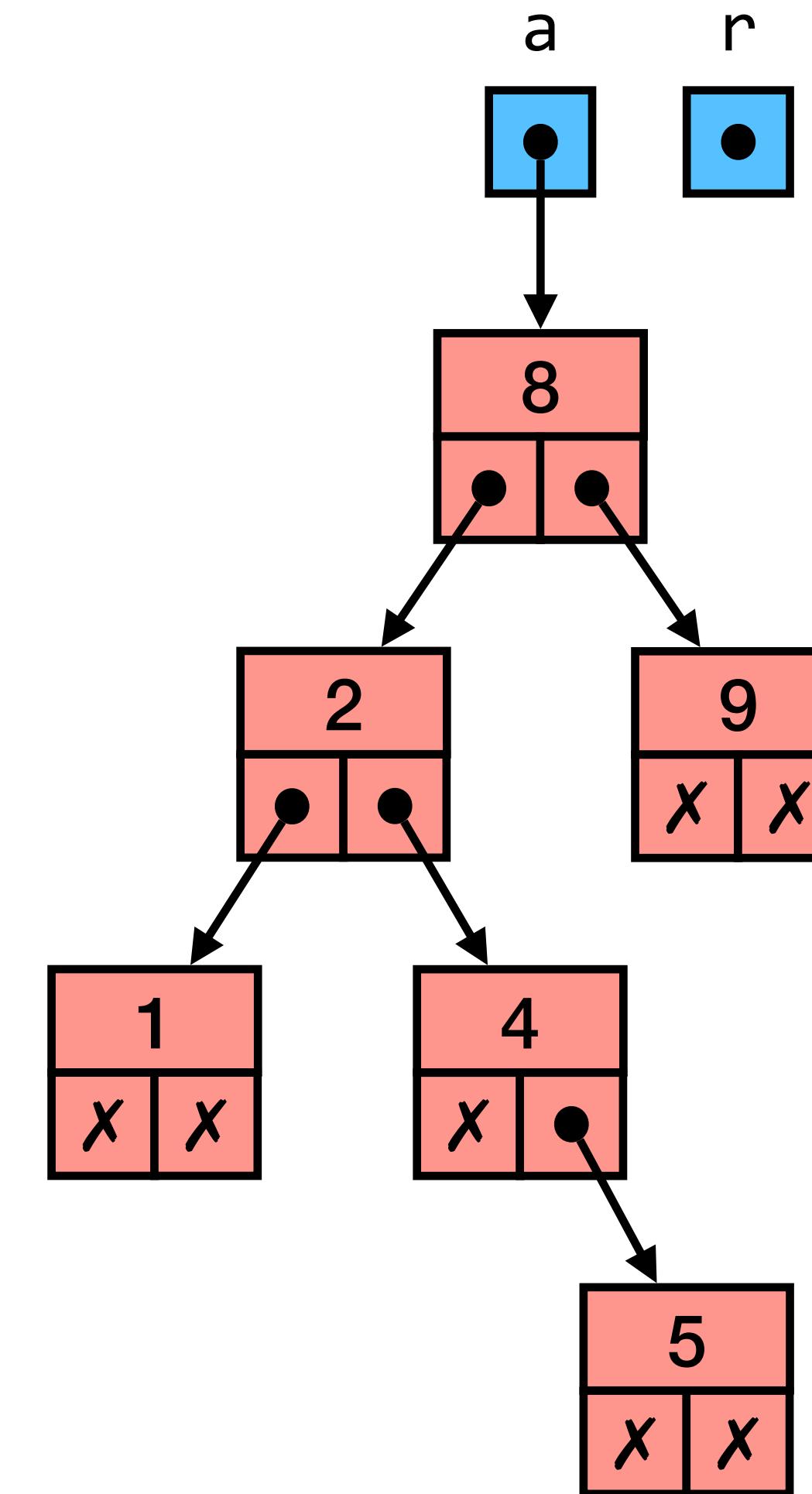
```
abin search(int x, abin a) {  
    if (a == NULL) return NULL;  
    if (a->valor == x) return a;  
    if (a->valor > x) return search(x, a->esq);  
    else return search(x, a->dir);  
}
```

Pesquisa ordenada

```
abin search(int x, abin a) {
    while (a != NULL && a->valor != x) {
        if (a->valor > x) a = a->esq;
        else a = a->dir;
    }
    return a;
}
```

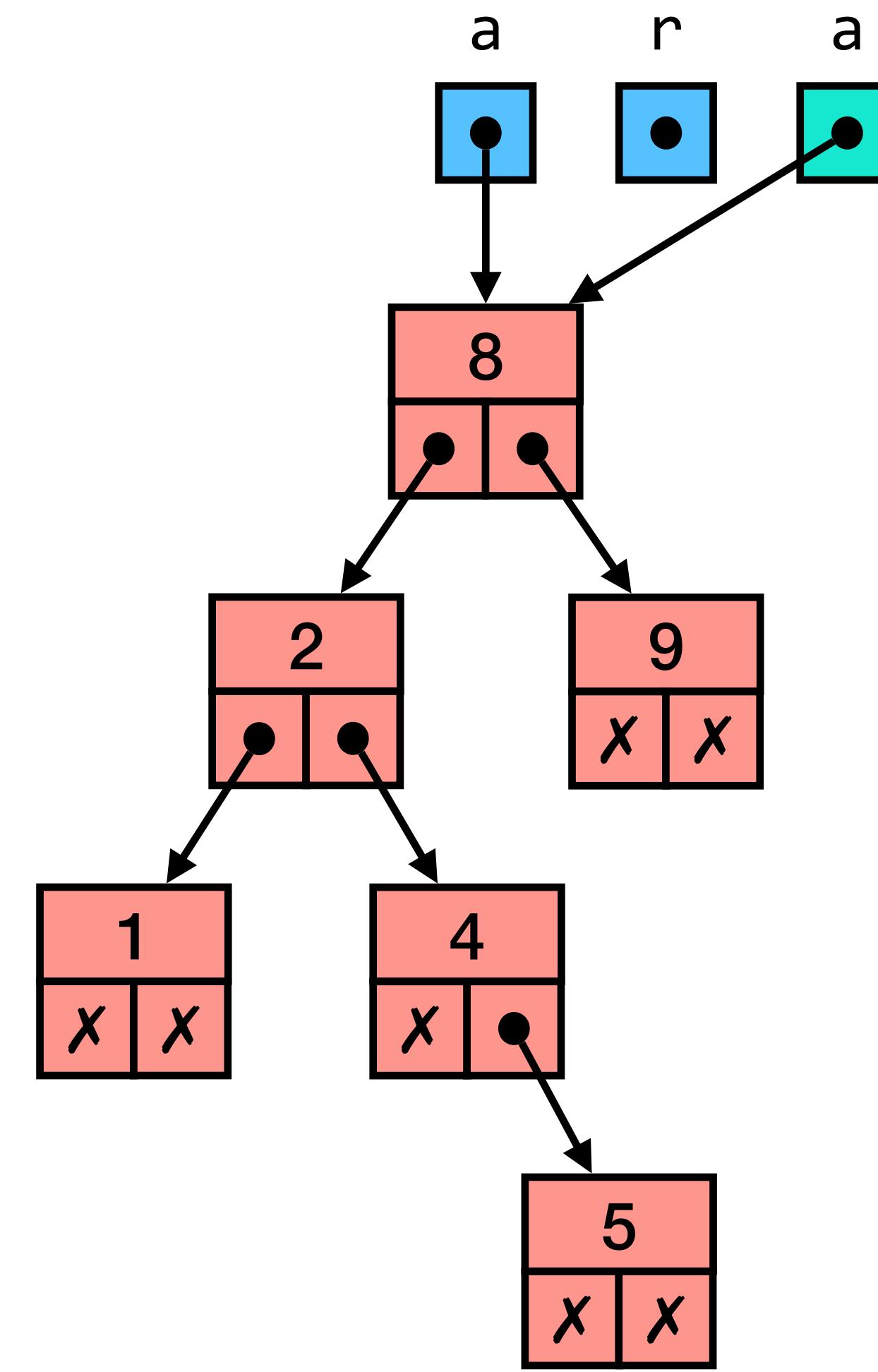
Pesquisa ordenada

```
abin search(int x, abin a) {  
    while (a != NULL && a->valor != x) {  
        if (a->valor > x) a = a->esq;  
        else a = a->dir;  
    }  
    return a;  
}  
  
int main() {  
    abin a = ...;  
    abin r = search(5, a);  
    return 0;  
}
```



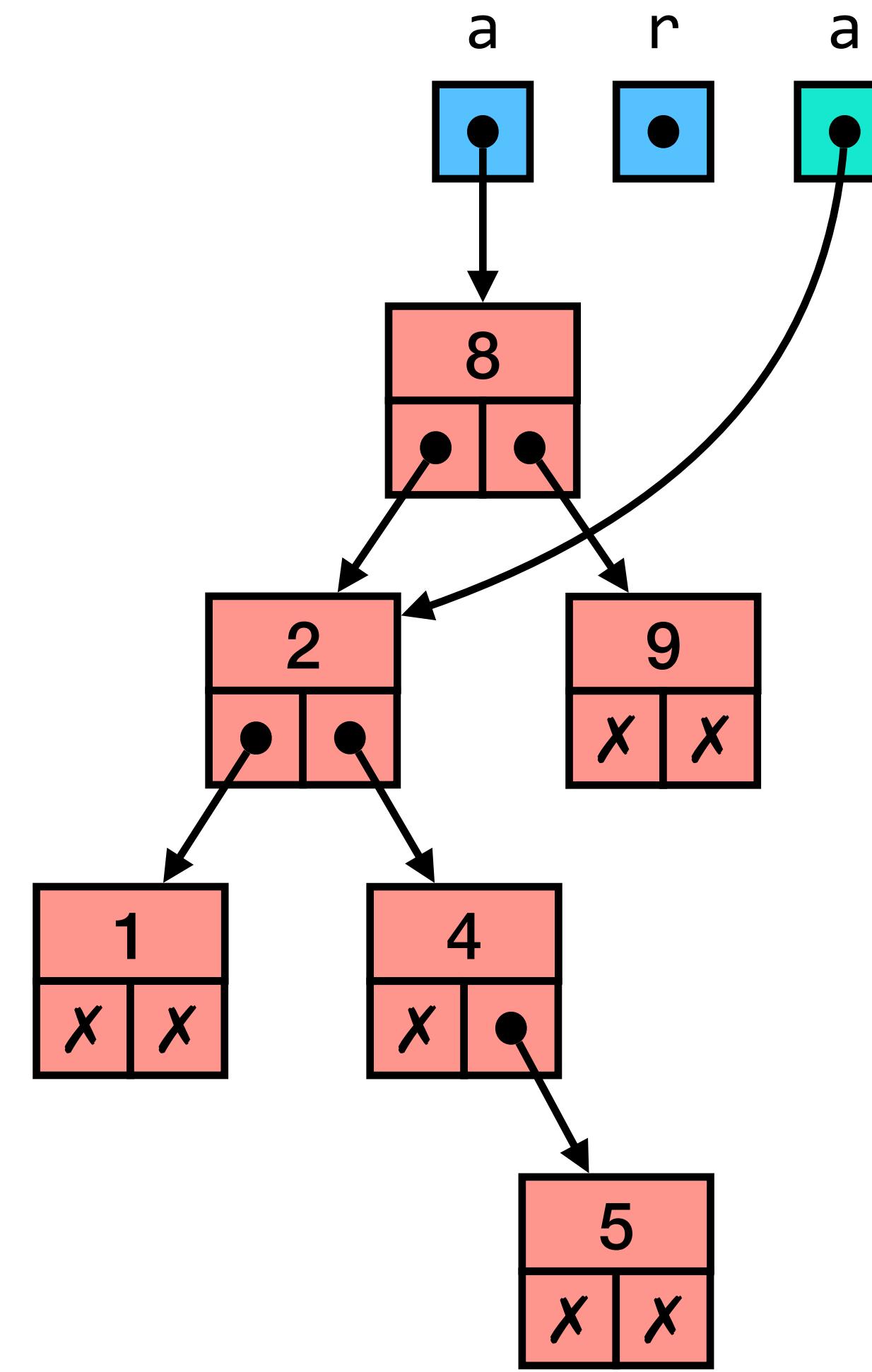
Pesquisa ordenada

```
abin search(int x, abin a) {  
    while (a != NULL && a->valor != x) {  
        if (a->valor > x) a = a->esq;  
        else a = a->dir;  
    }  
    return a;  
}  
  
int main() {  
    abin a = ...;  
    abin r = search(5, a);  
    return 0;  
}
```



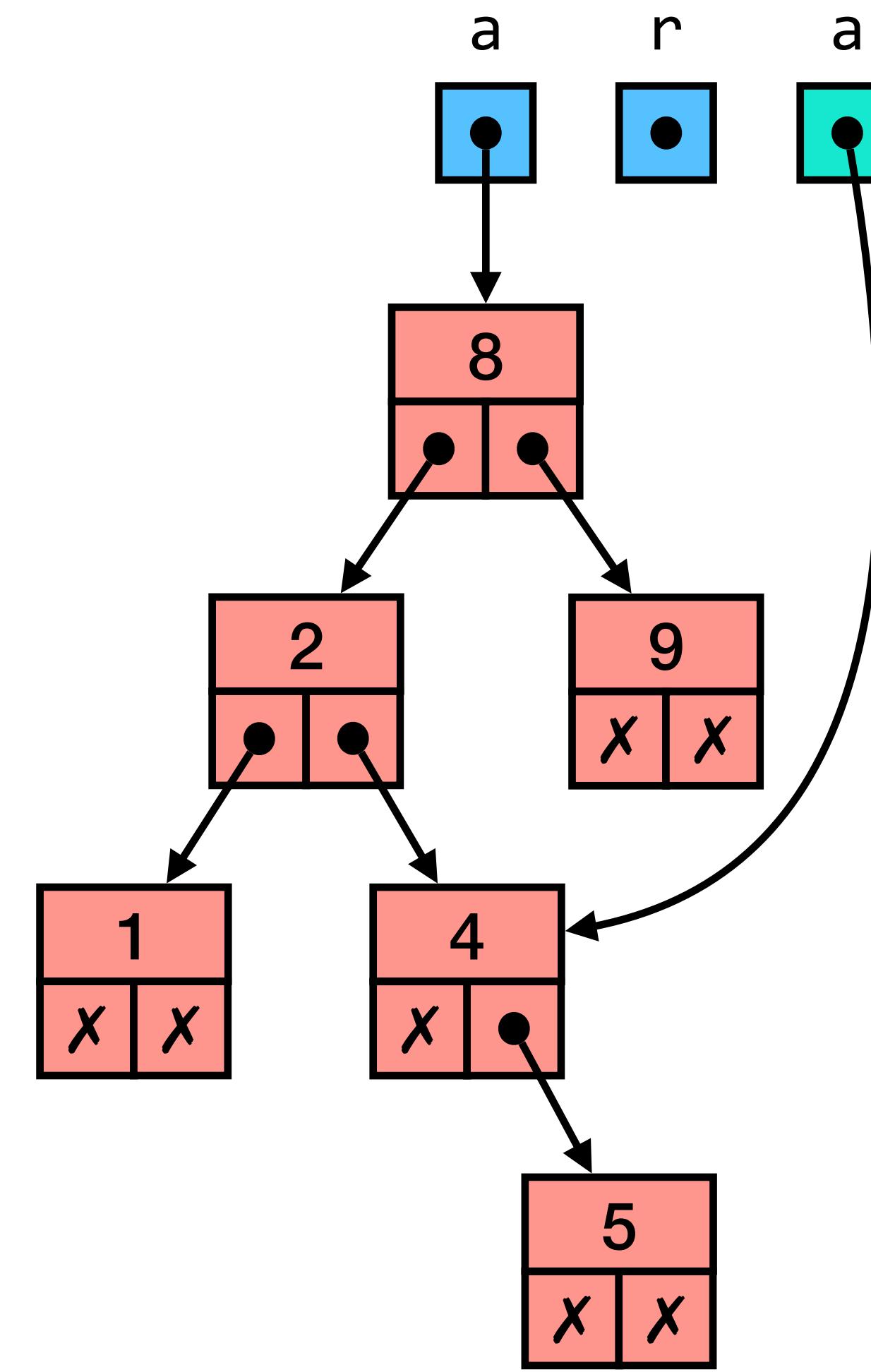
Pesquisa ordenada

```
abin search(int x, abin a) {  
    while (a != NULL && a->valor != x) {  
        if (a->valor > x) a = a->esq;  
        else a = a->dir;  
    }  
    return a;  
}  
  
int main() {  
    abin a = ...;  
    abin r = search(5, a);  
    return 0;  
}
```



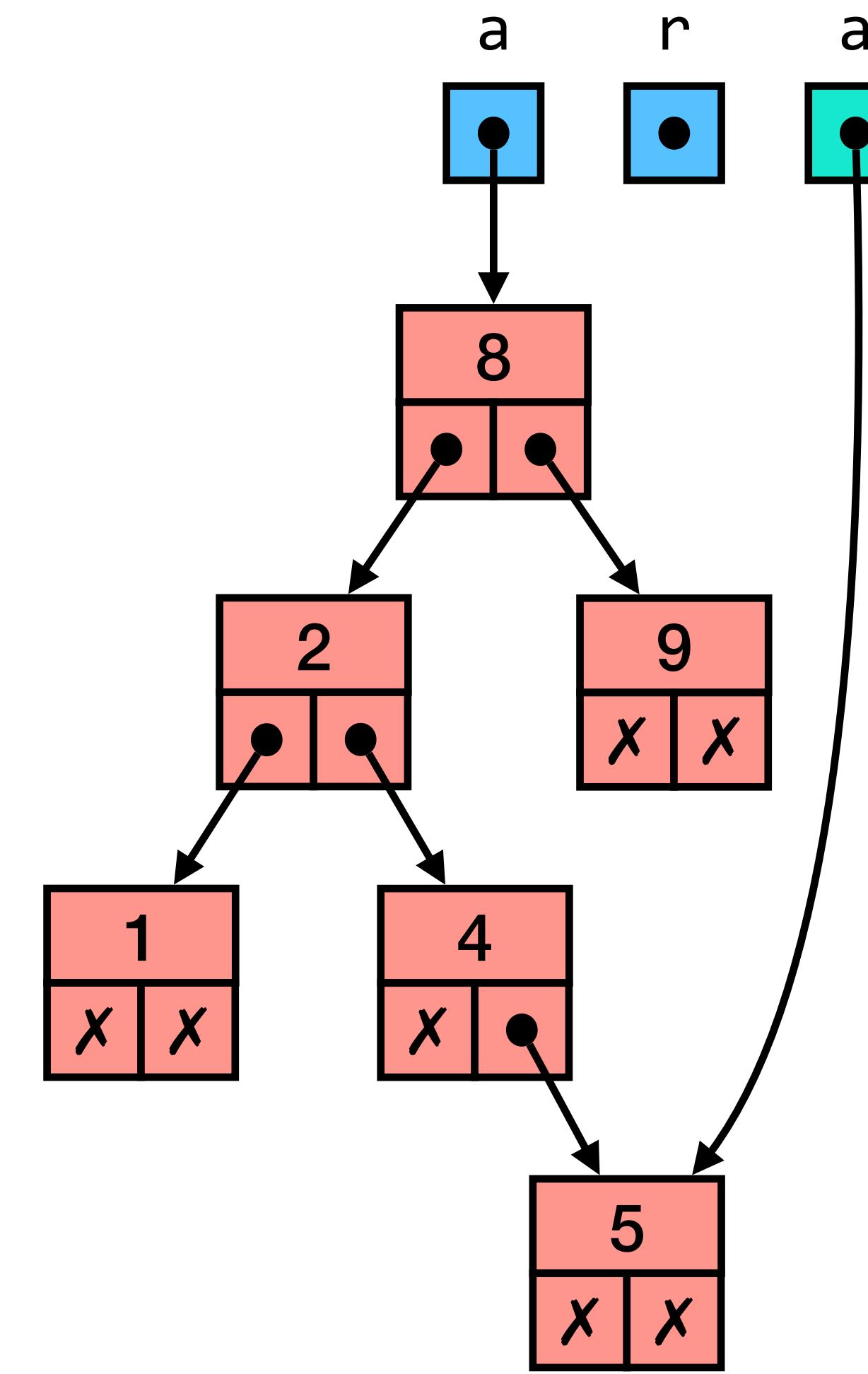
Pesquisa ordenada

```
abin search(int x, abin a) {  
    while (a != NULL && a->valor != x) {  
        if (a->valor > x) a = a->esq;  
        else a = a->dir;  
    }  
    return a;  
}  
  
int main() {  
    abin a = ...;  
    abin r = search(5, a);  
    return 0;  
}
```



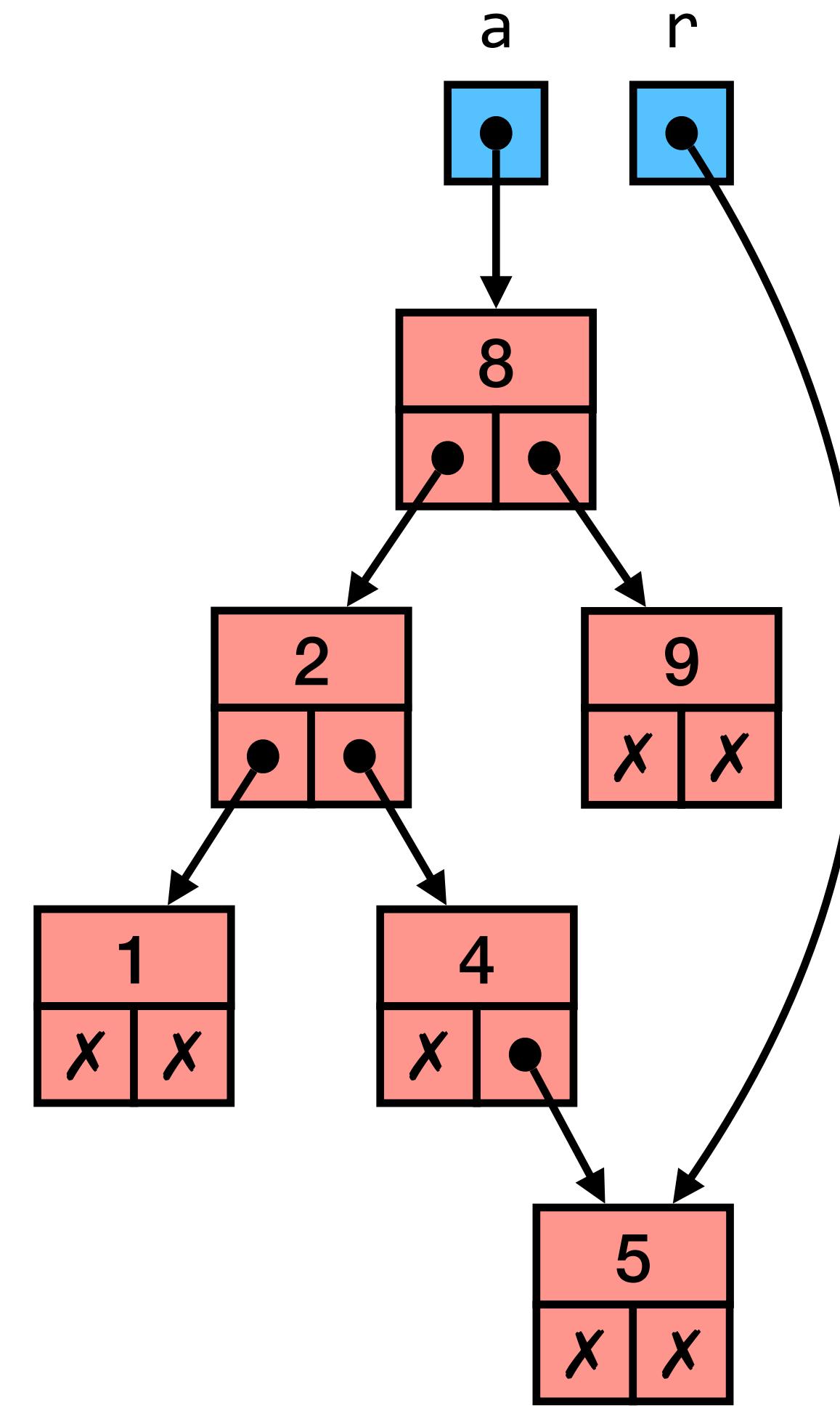
Pesquisa ordenada

```
abin search(int x, abin a) {  
    while (a != NULL && a->valor != x) {  
        if (a->valor > x) a = a->esq;  
        else a = a->dir;  
    }  
    return a;  
}  
  
int main() {  
    abin a = ...;  
    abin r = search(5, a);  
    return 0;  
}
```



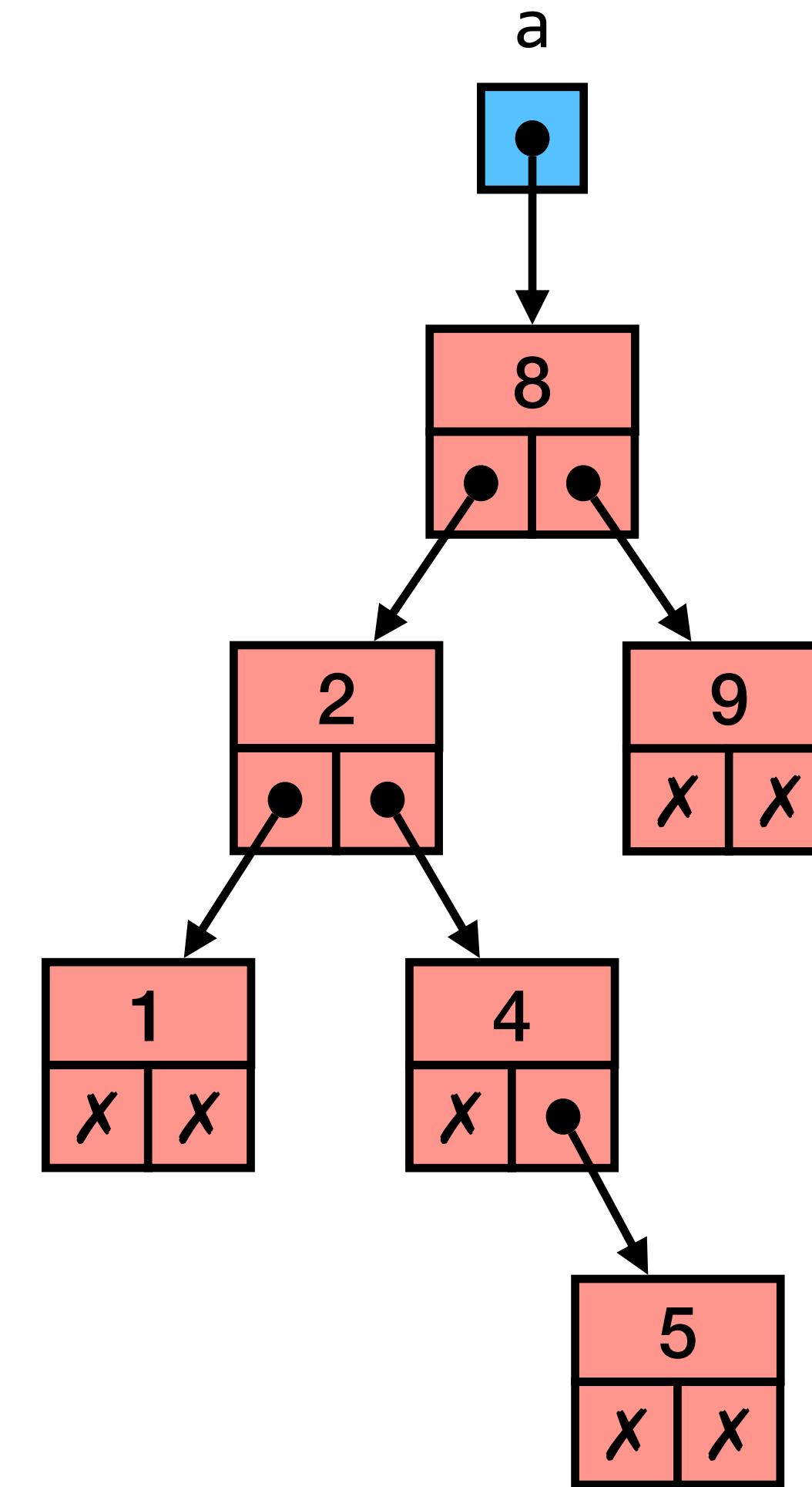
Pesquisa ordenada

```
abin search(int x, abin a) {  
    while (a != NULL && a->valor != x) {  
        if (a->valor > x) a = a->esq;  
        else a = a->dir;  
    }  
    return a;  
}  
  
int main() {  
    abin a = ...;  
    abin r = search(5, a);  
    return 0;  
}
```



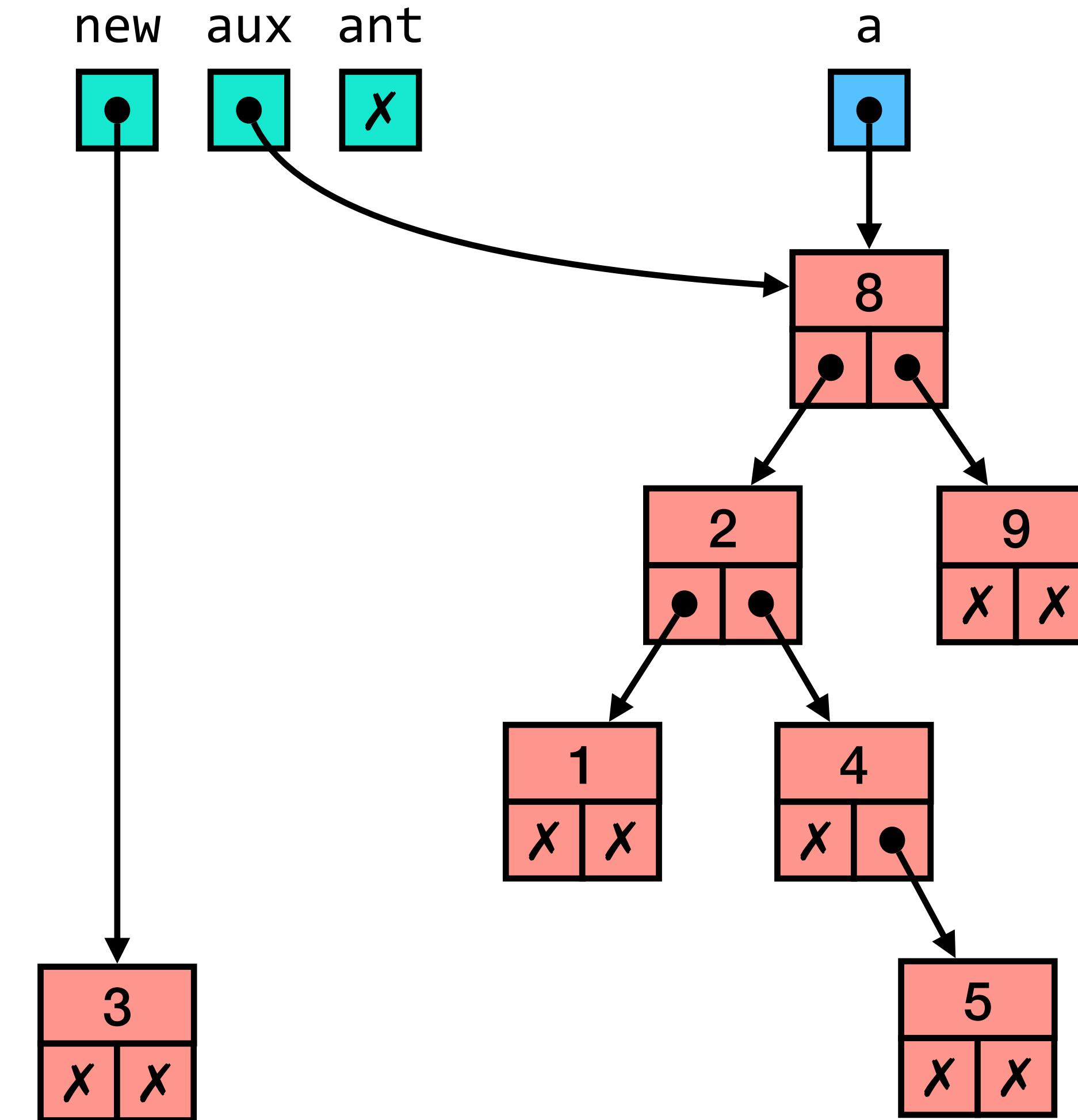
Inserção ordenada

```
int main() {
    abin a = ...;
    a = insert(3, a);
    return 0;
}
```



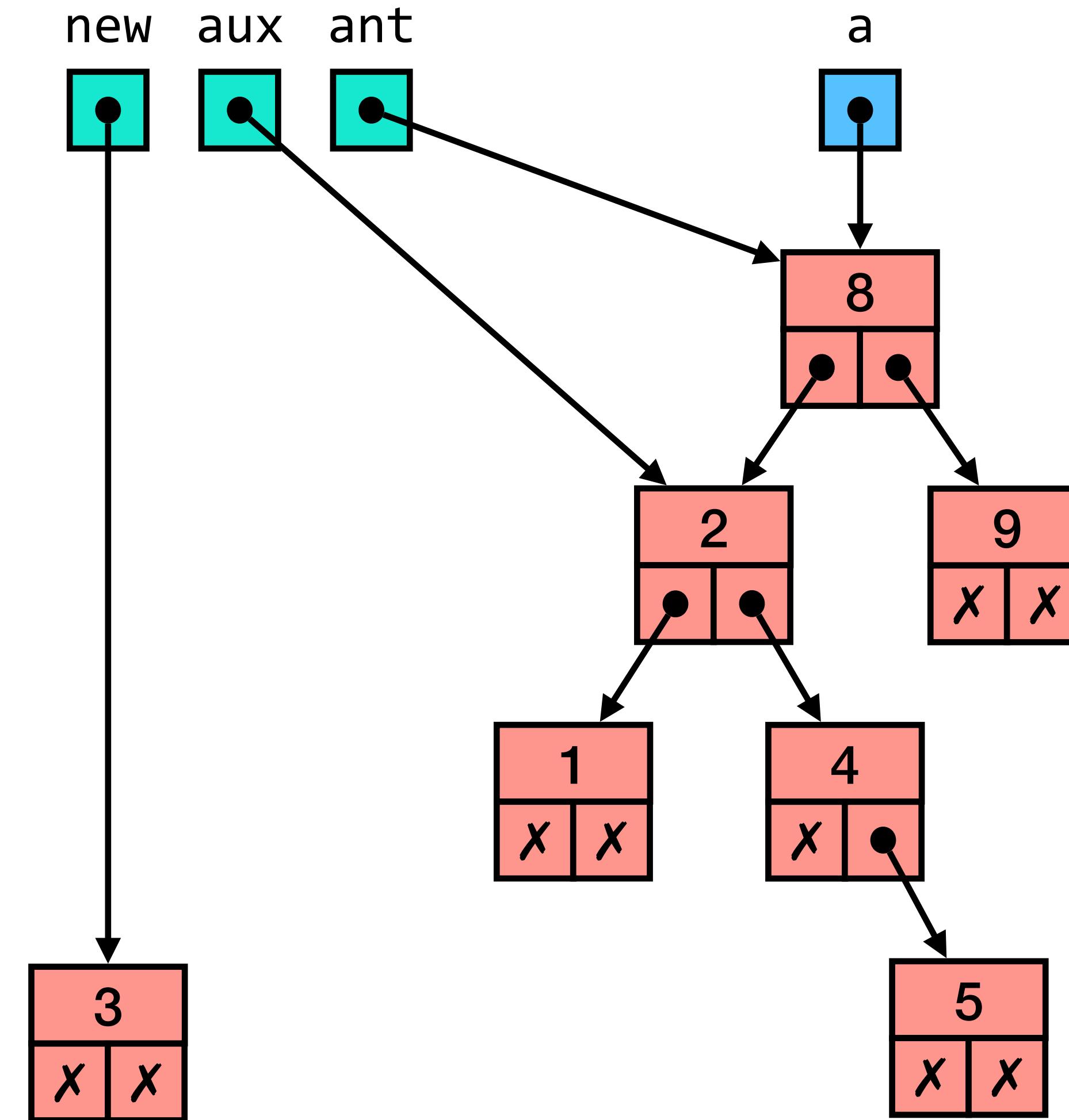
Inserção ordenada

```
int main() {  
    abin a = ...;  
    a = insert(3, a);  
    return 0;  
}
```



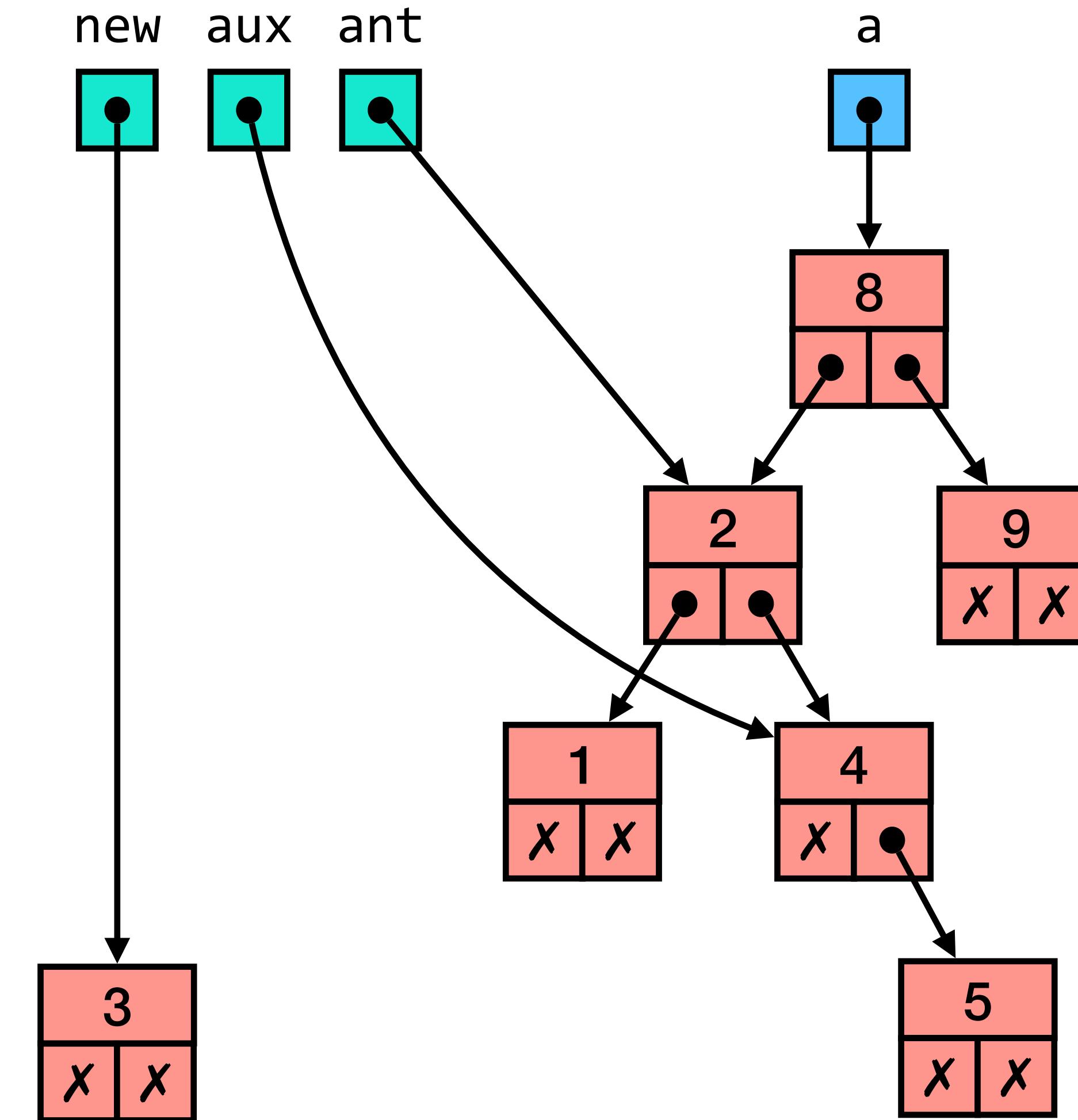
Inserção ordenada

```
int main() {  
    abin a = ...;  
    a = insert(3, a);  
    return 0;  
}
```



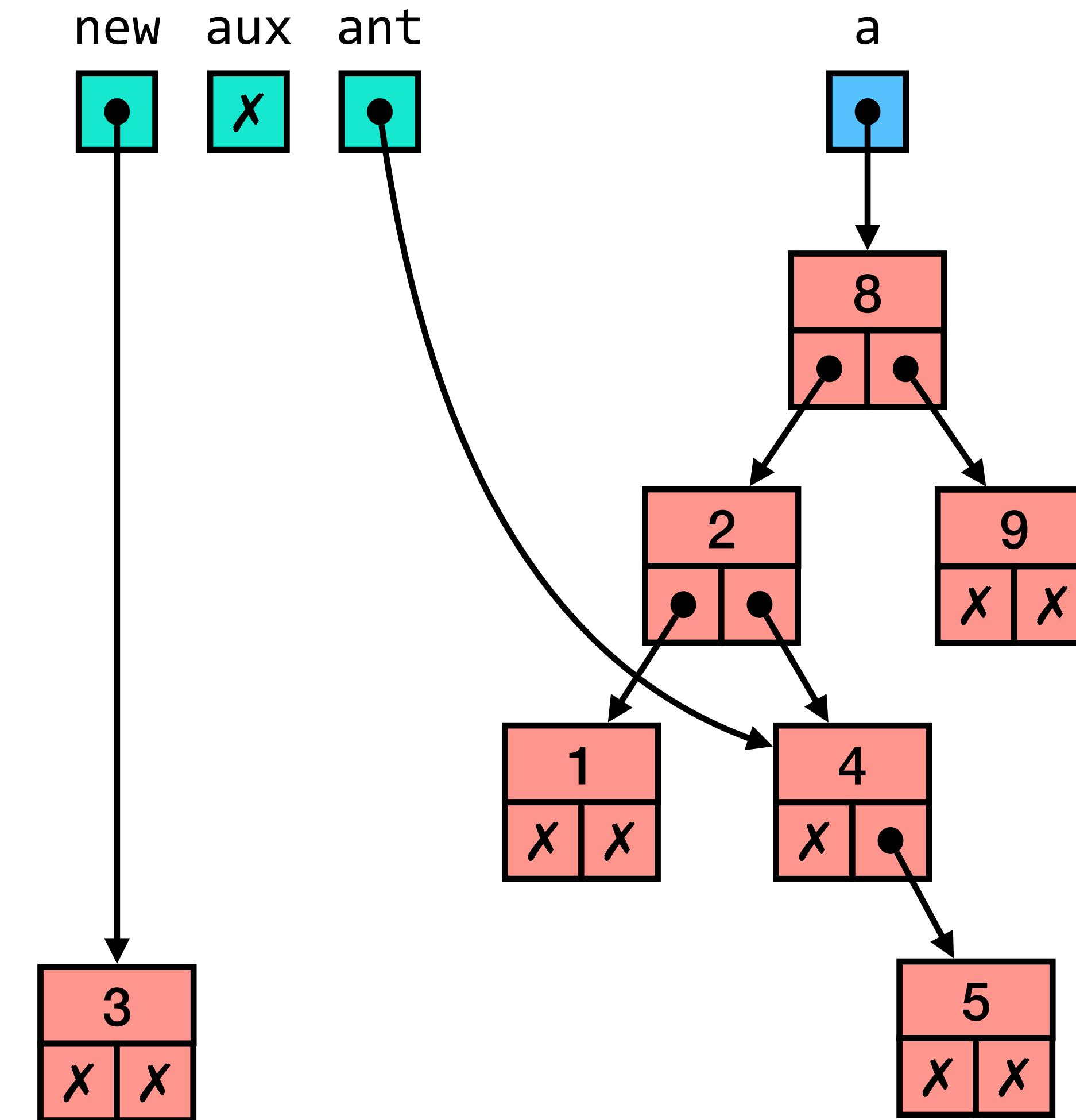
Inserção ordenada

```
int main() {  
    abin a = ...;  
    a = insert(3, a);  
    return 0;  
}
```



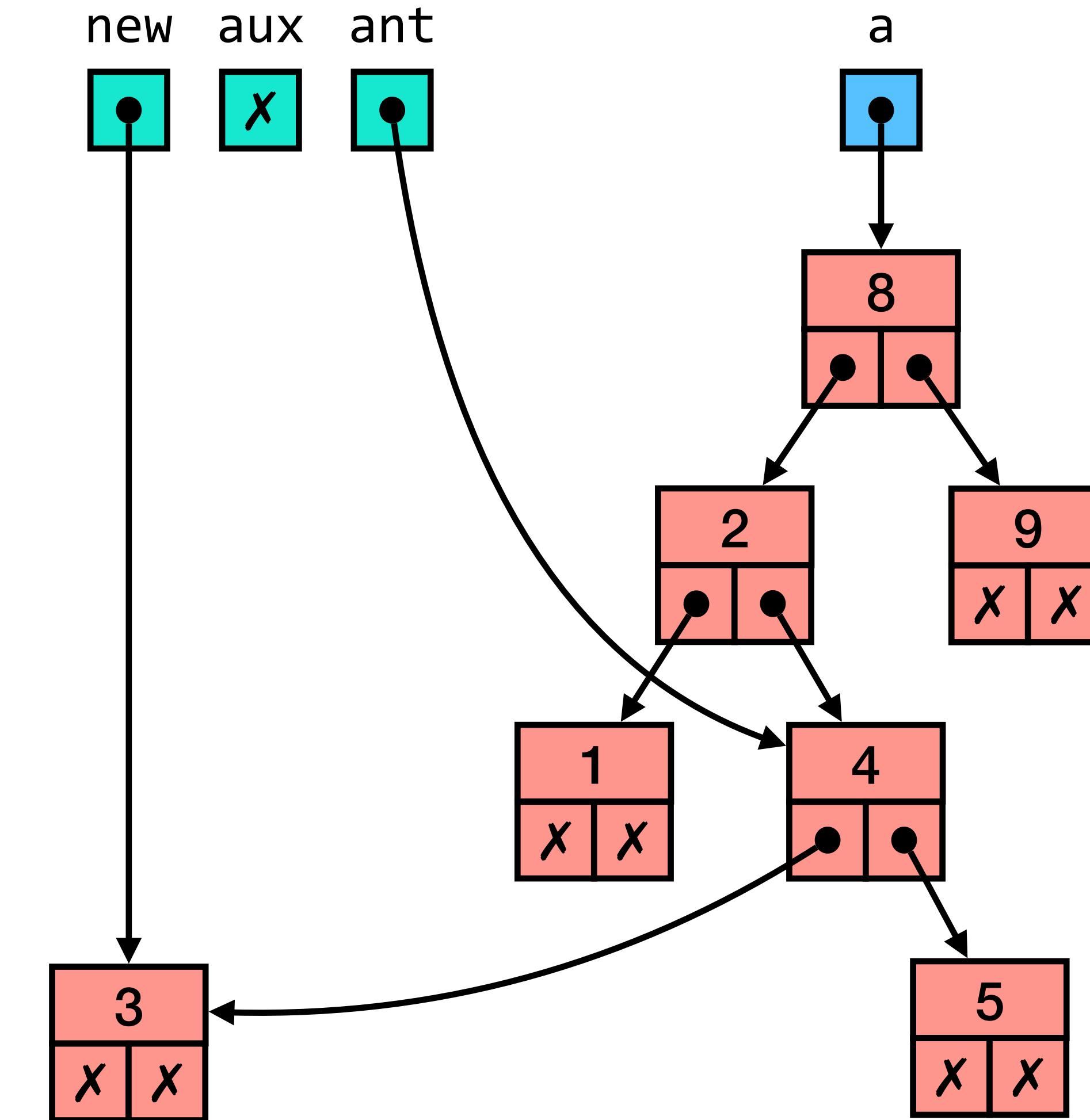
Inserção ordenada

```
int main() {  
    abin a = ...;  
    a = insert(3, a);  
    return 0;  
}
```



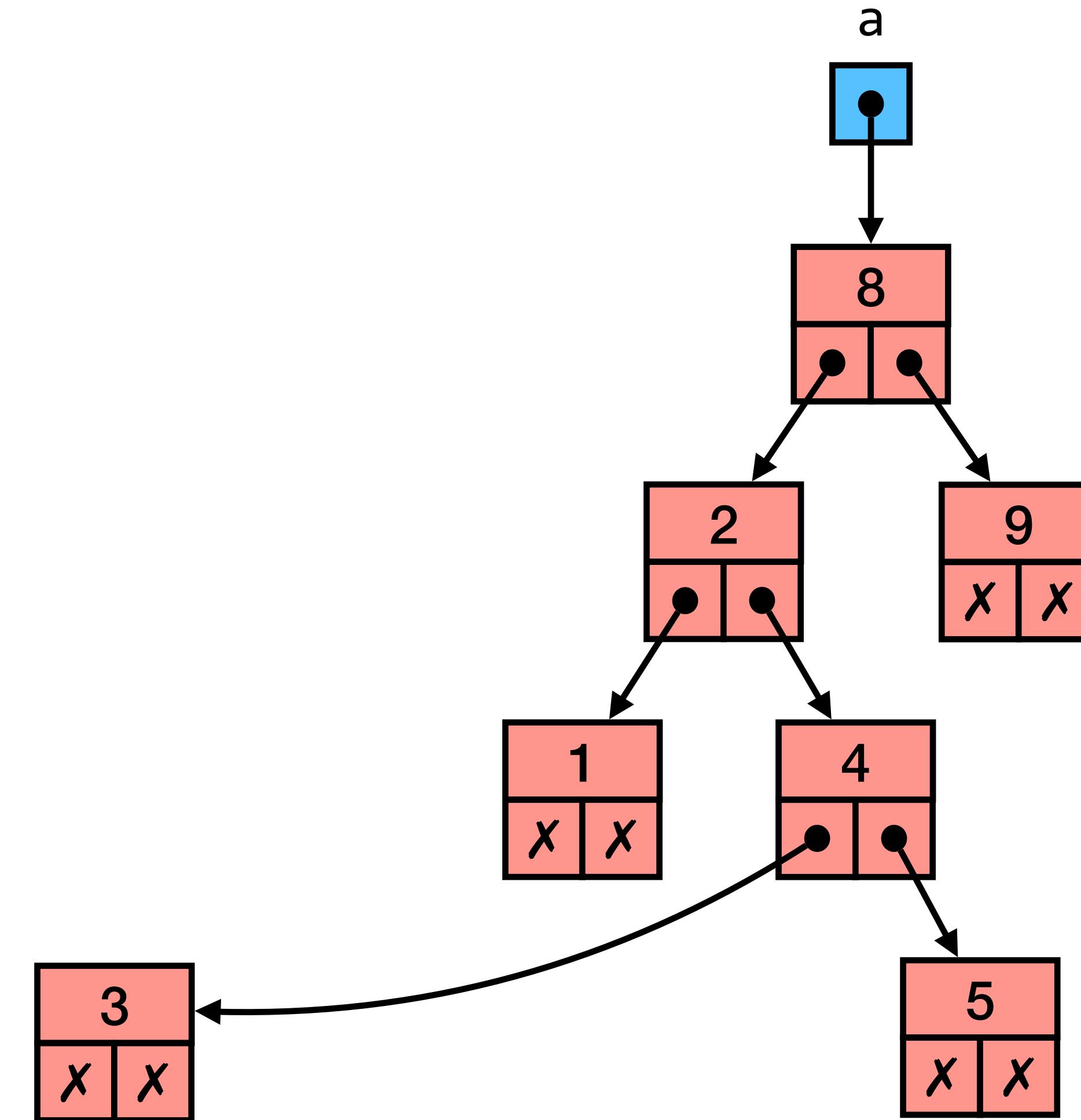
Inserção ordenada

```
int main() {  
    abin a = ...;  
    a = insert(3, a);  
    return 0;  
}
```



Inserção ordenada

```
int main() {
    abin a = ...;
    a = insert(3, a);
    return 0;
}
```

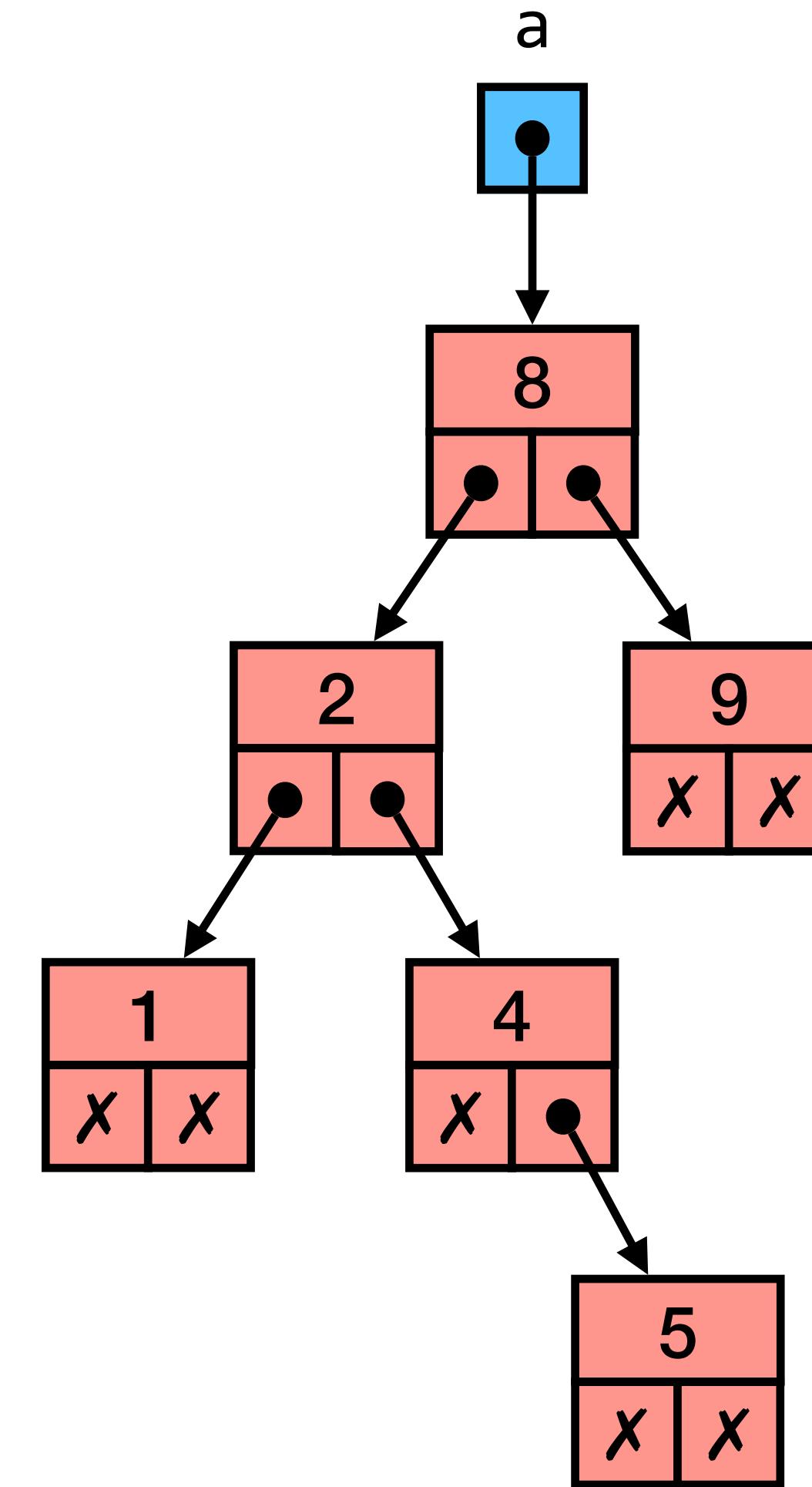


Inserção ordenada

```
abin insert(int x, abin a) {
    abin ant = NULL, aux = a, new = mkroot(x, NULL, NULL);
    while (aux != NULL) {
        ant = aux;
        if (aux->valor > x) aux = aux->esq;
        else aux = aux->dir;
    }
    if (ant == NULL) a = new;
    else if (ant->valor > x) ant->esq = new;
    else ant->dir = new;
    return a;
}
```

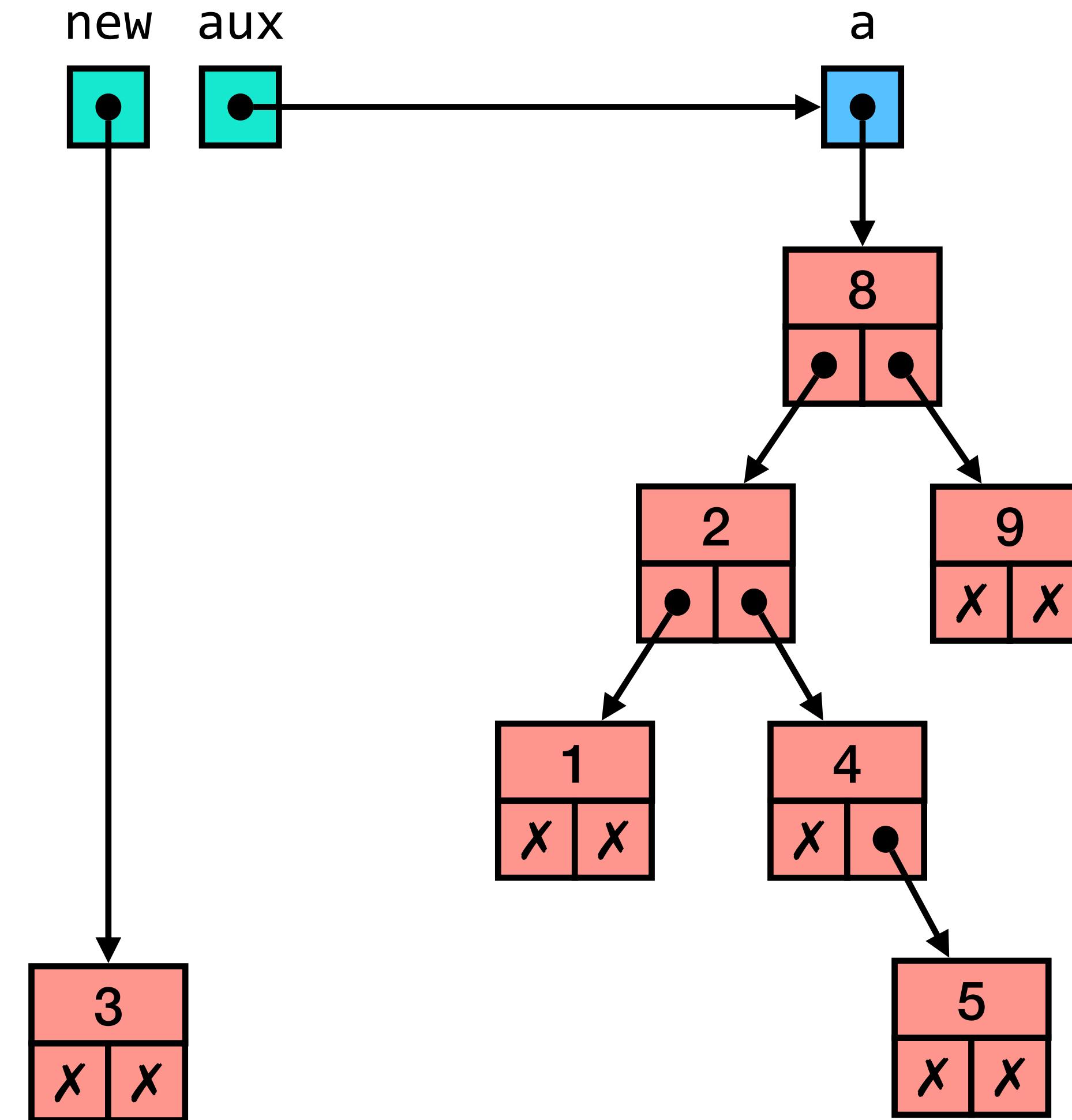
Inserção ordenada

```
int main() {
    abin a = ...;
    a = insert(3, a);
    return 0;
}
```



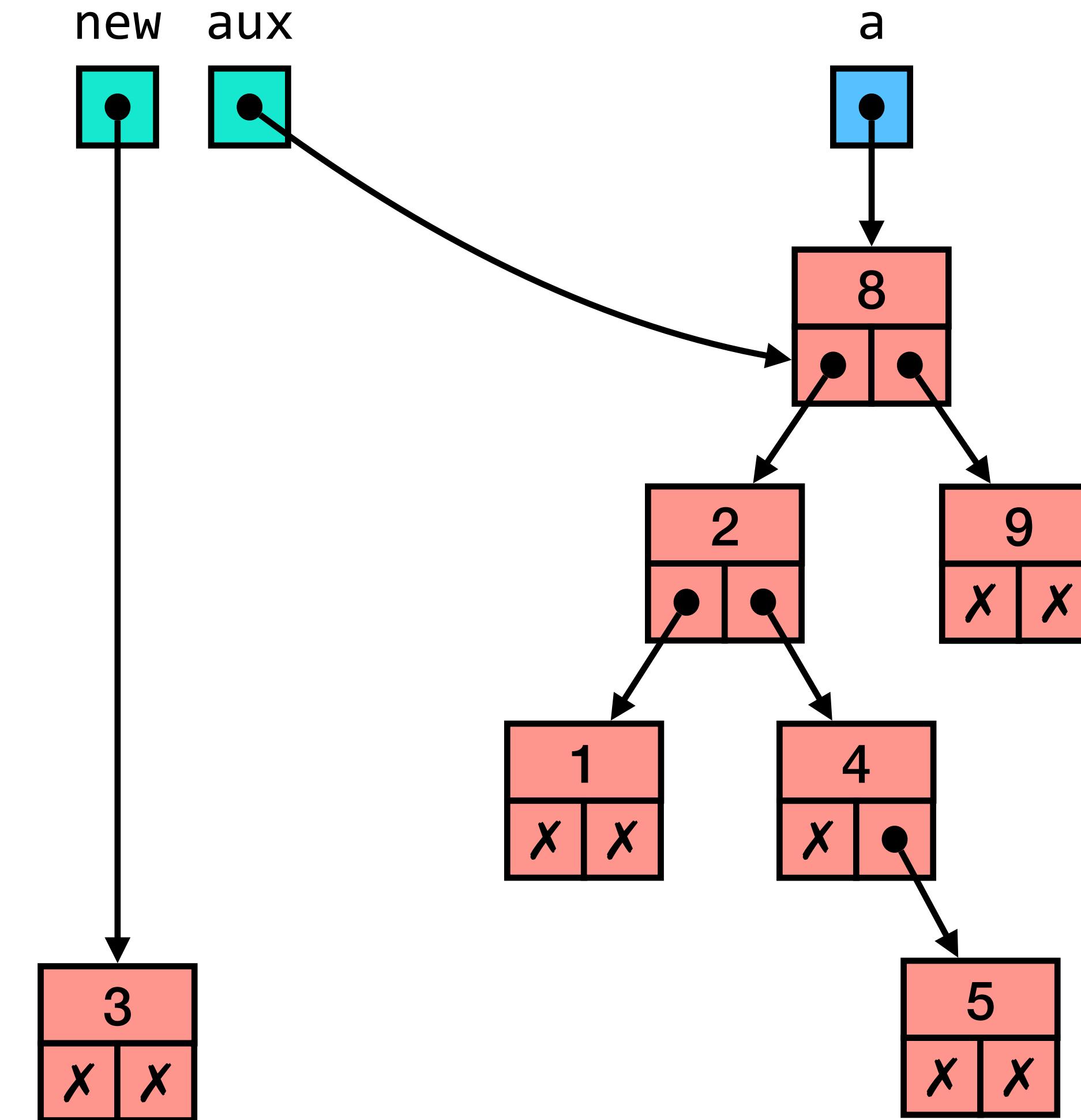
Inserção ordenada

```
int main() {
    abin a = ...;
    a = insert(3, a);
    return 0;
}
```



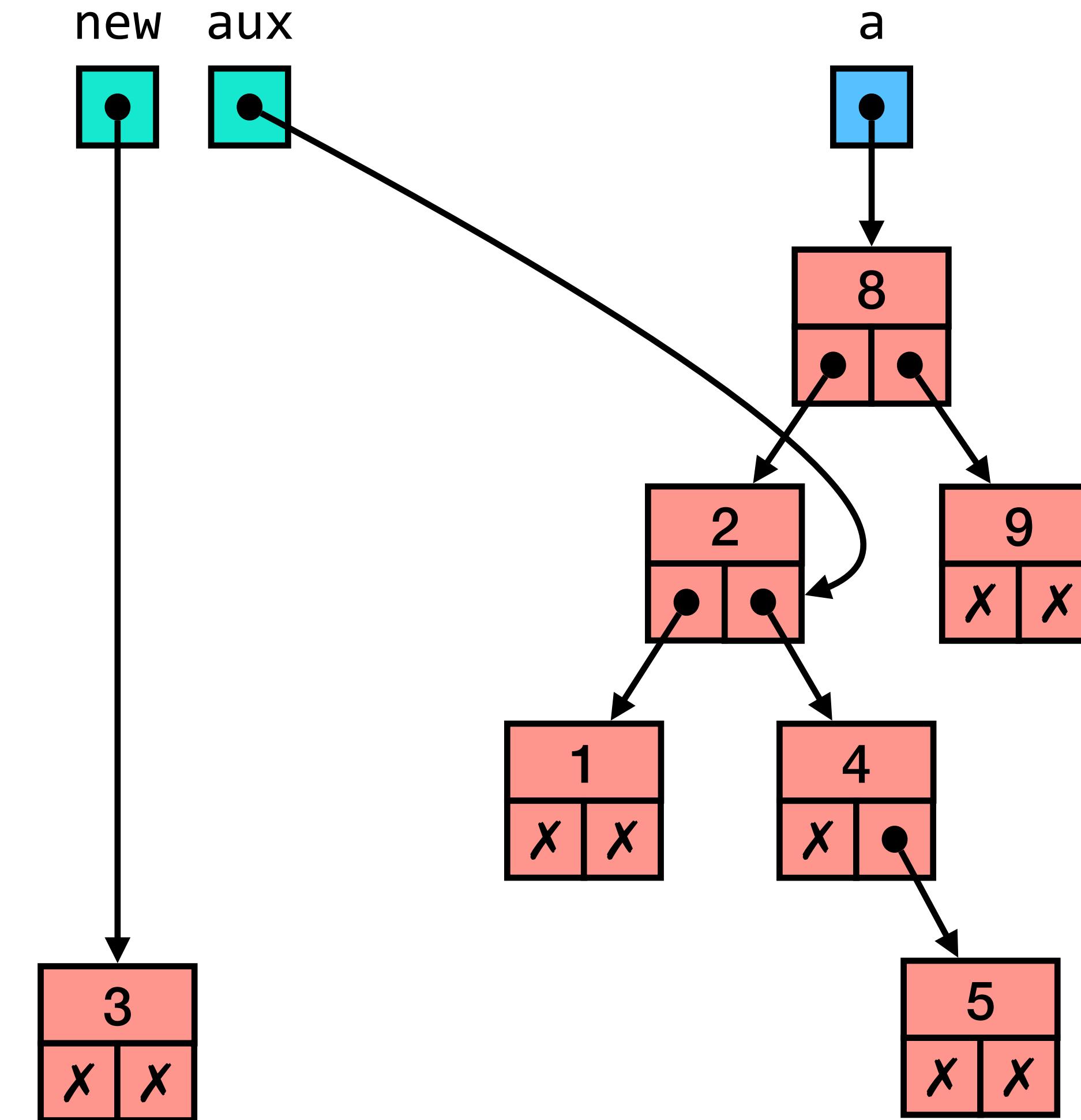
Inserção ordenada

```
int main() {
    abin a = ...;
    a = insert(3, a);
    return 0;
}
```



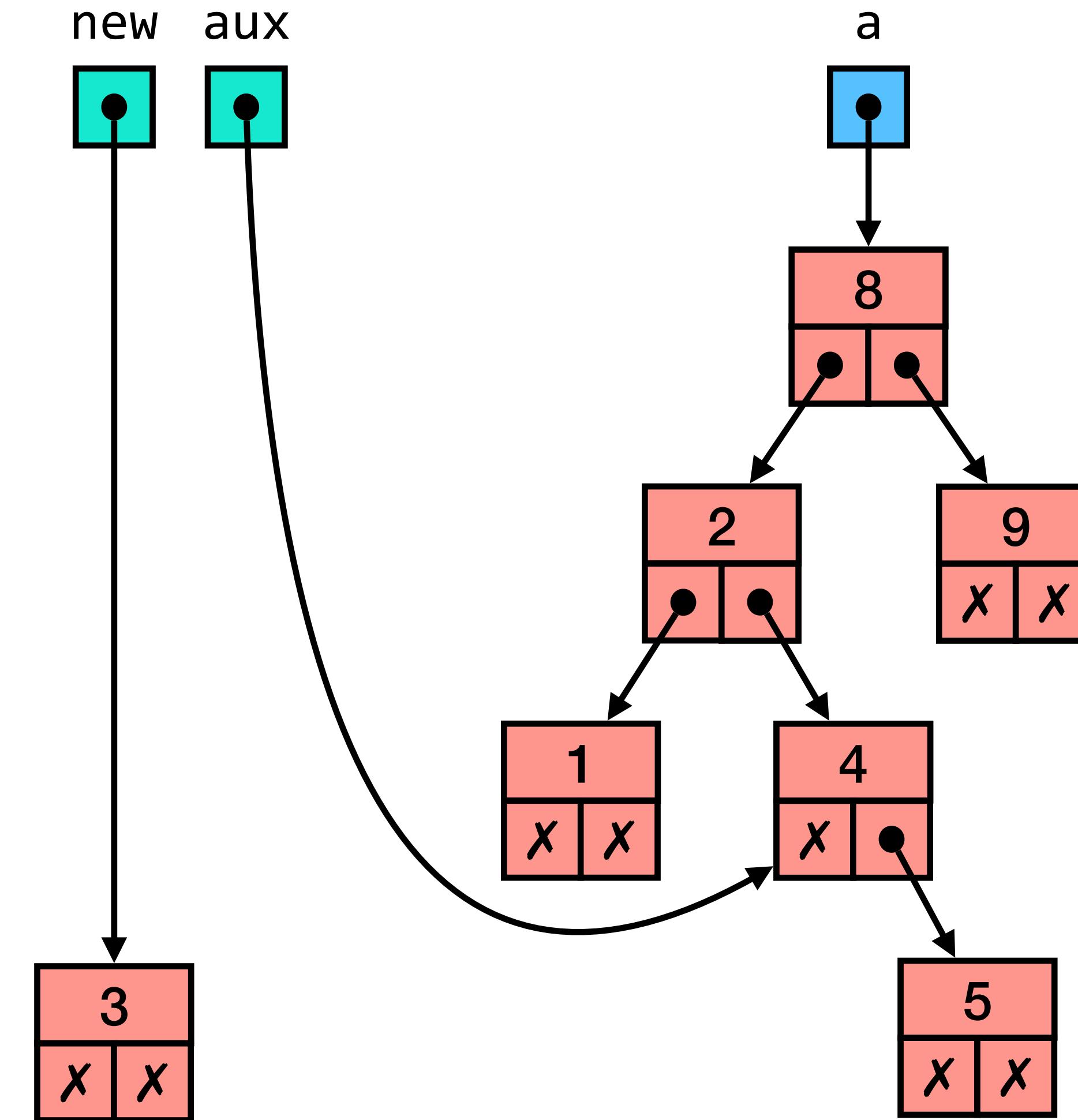
Inserção ordenada

```
int main() {
    abin a = ...;
    a = insert(3, a);
    return 0;
}
```



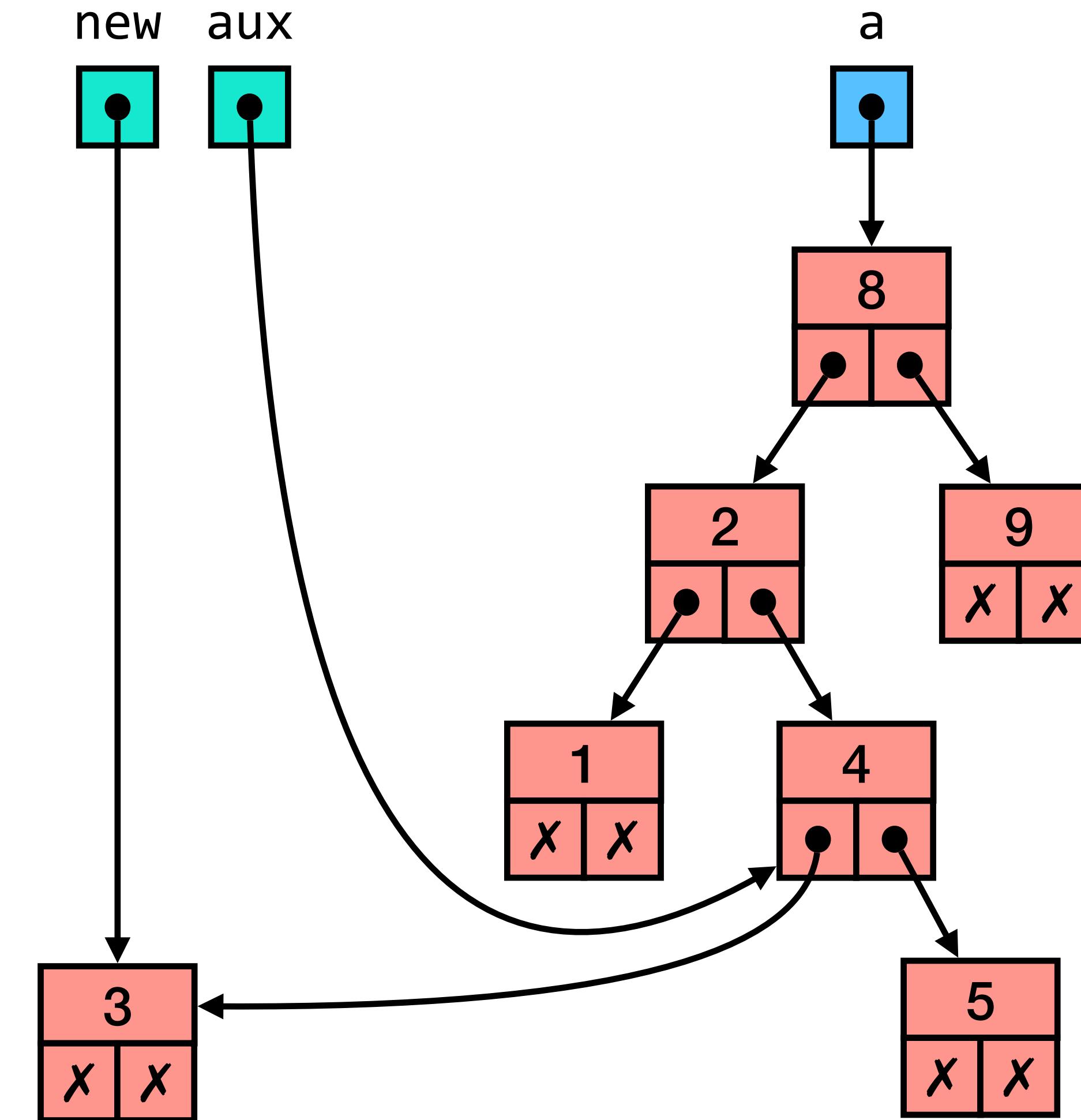
Inserção ordenada

```
int main() {  
    abin a = ...;  
    a = insert(3, a);  
    return 0;  
}
```



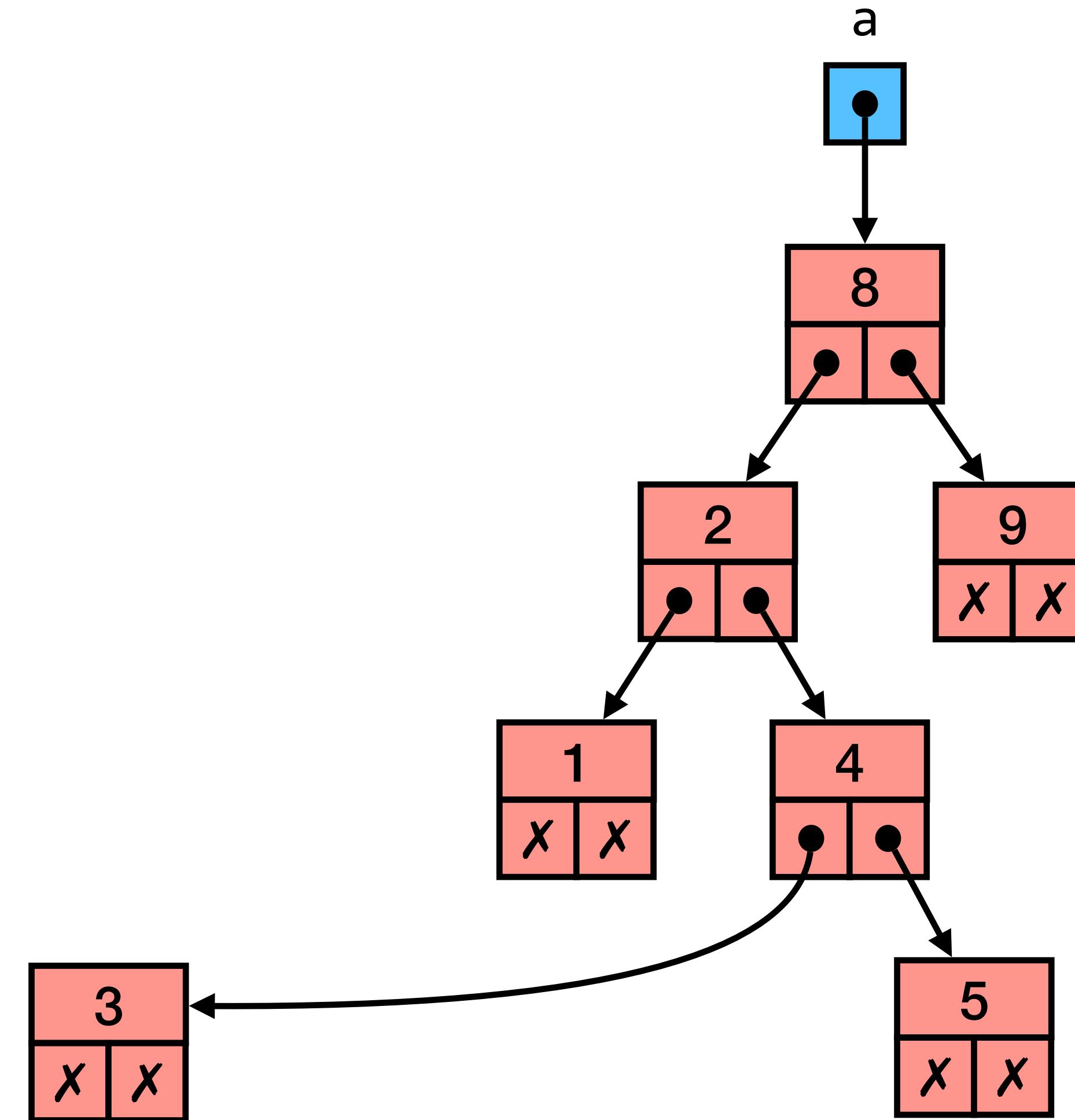
Inserção ordenada

```
int main() {
    abin a = ...;
    a = insert(3, a);
    return 0;
}
```



Inserção ordenada

```
int main() {
    abin a = ...;
    a = insert(3, a);
    return 0;
}
```

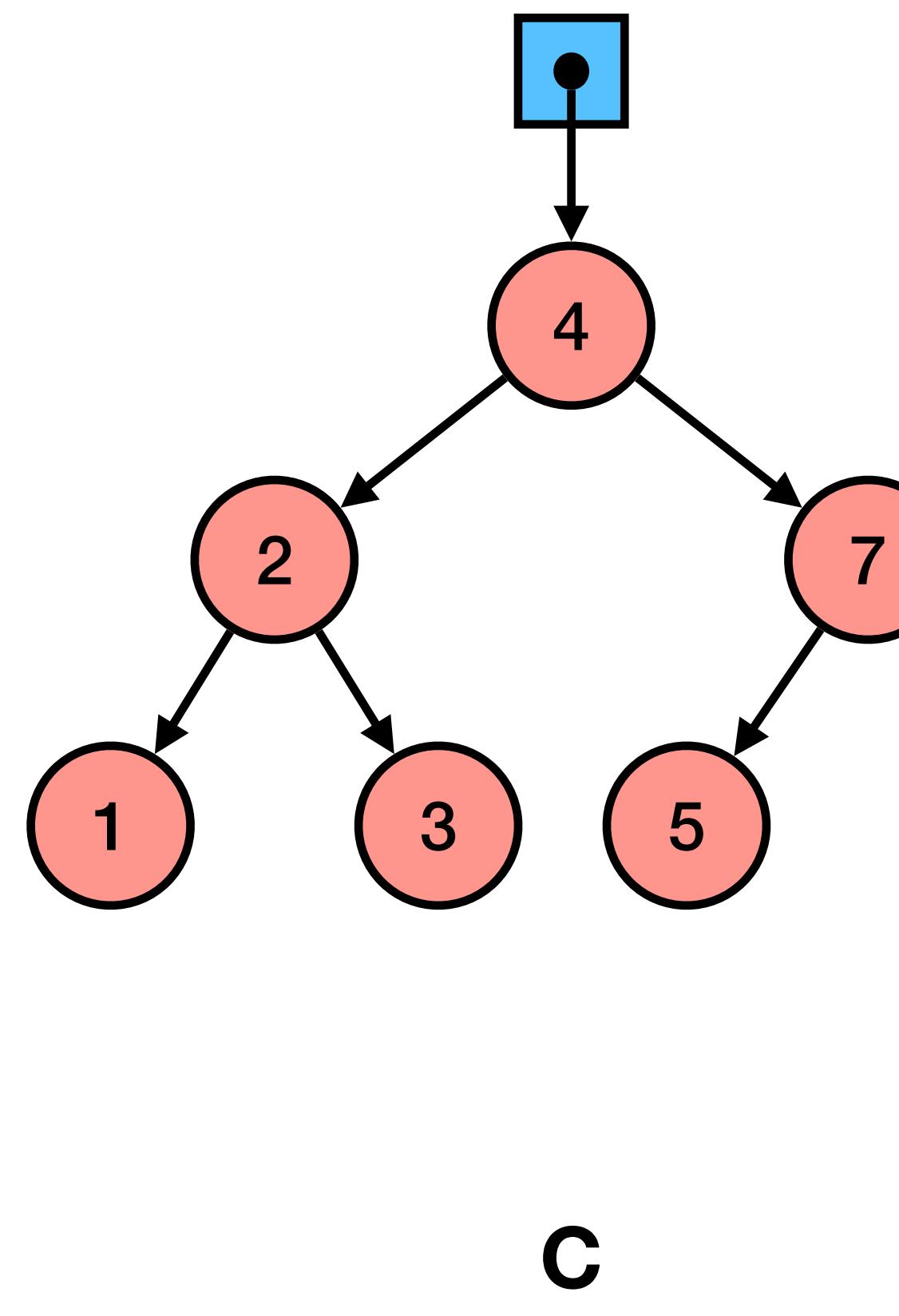
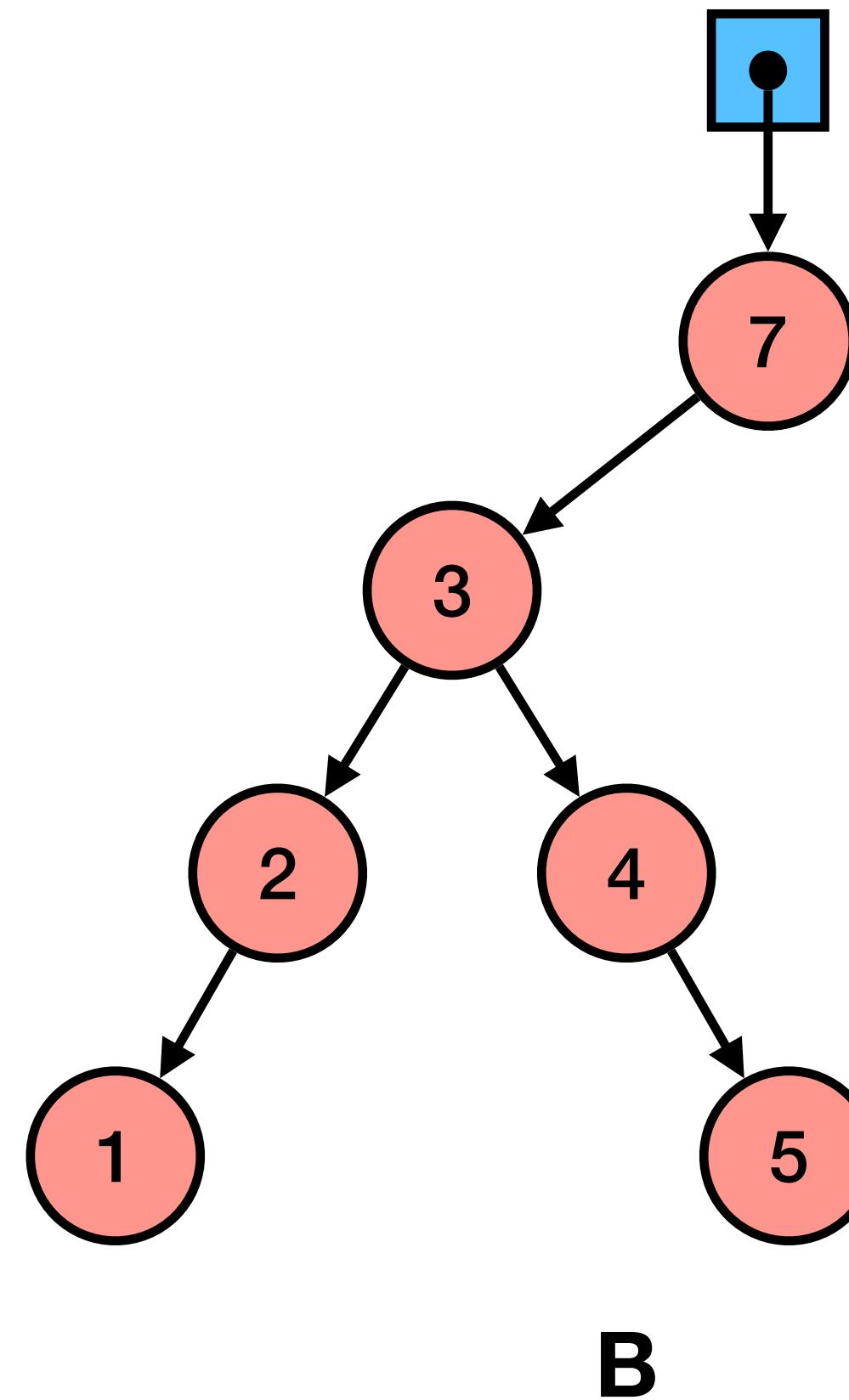
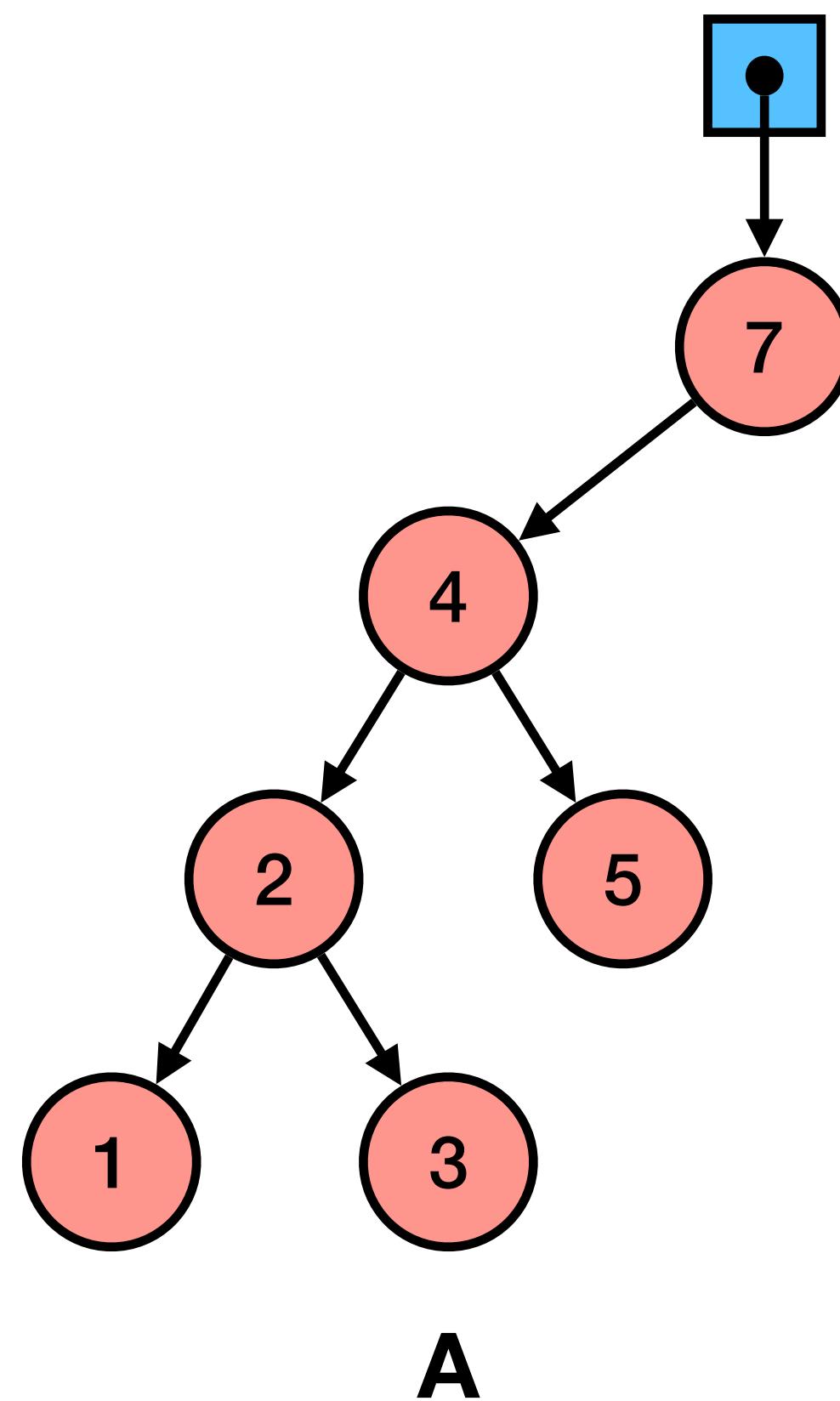


Inserção ordenada

```
abin insert(int x, abin a) {
    abin *aux = &a, new = mkroot(x, NULL, NULL);
    while (*aux != NULL) {
        if ((*aux)->valor > x) aux = &(*aux)->esq;
        else aux = &(*aux)->dir;
    }
    *aux = new;
    return a;
}
```

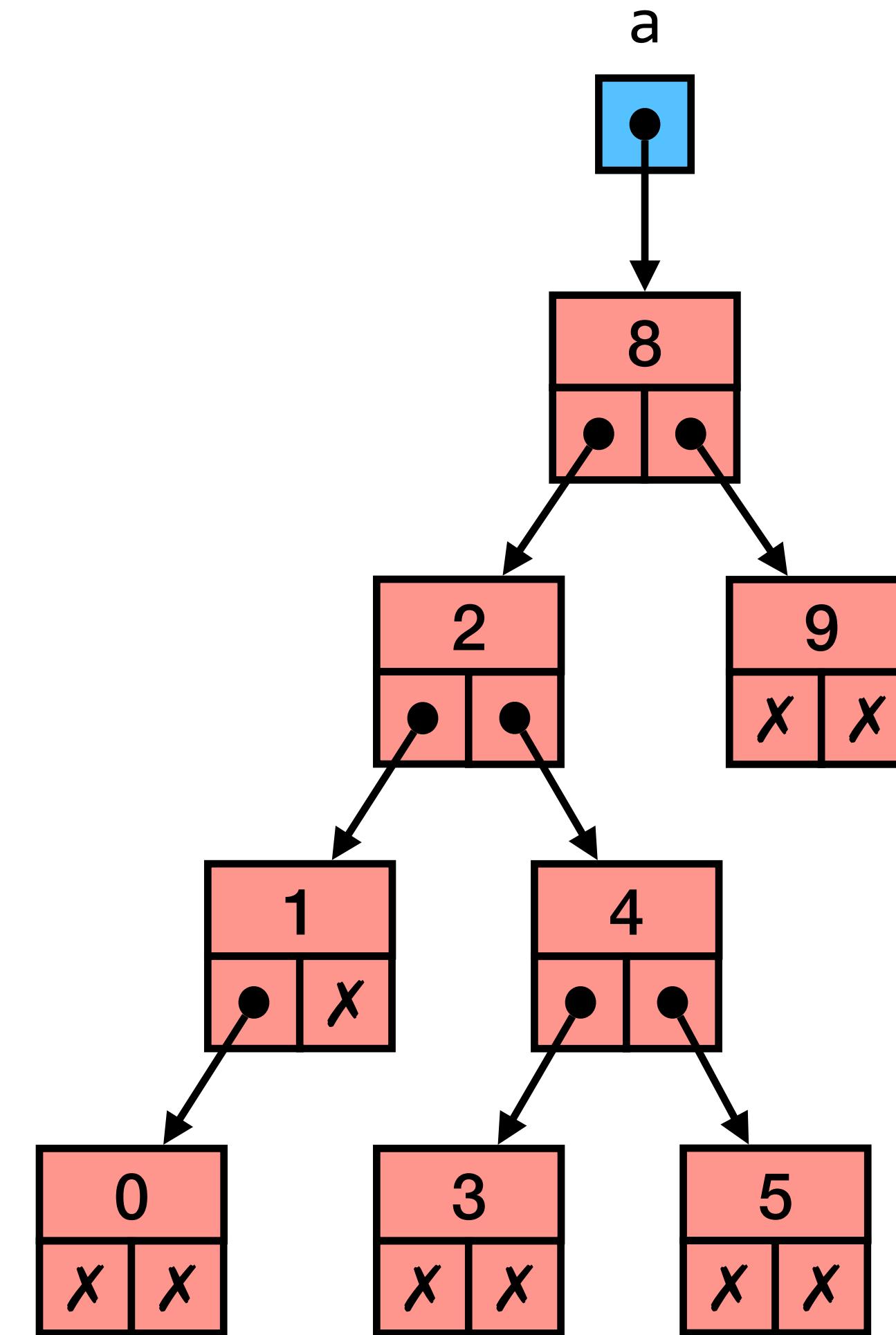
#30 Que árvore de procura vai ser criada?

```
int v[] = {7,3,4,5,2,1}; abin a = NULL;  
for (int i = 0; i < 6; i++) a = insert(v[i], a);
```



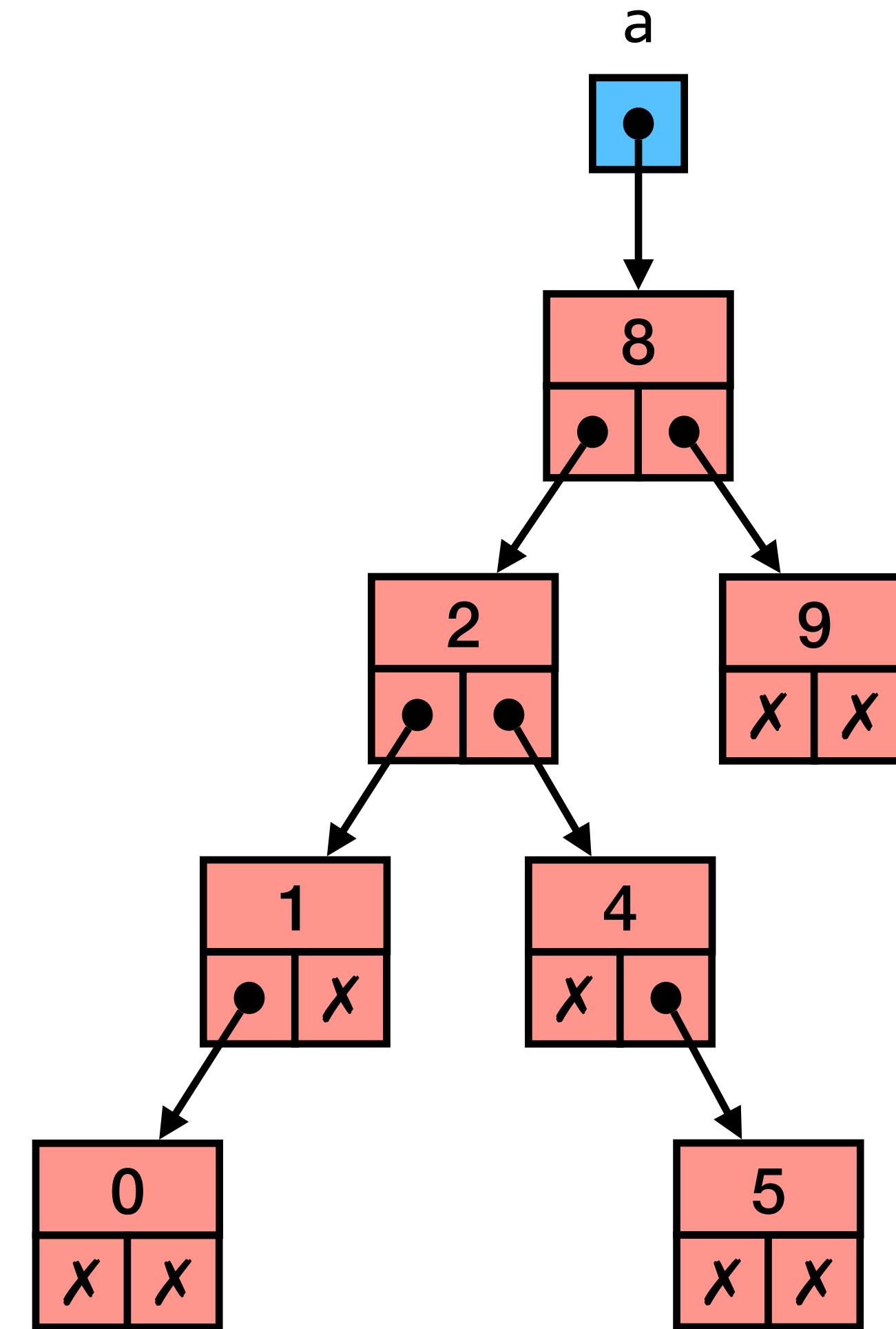
Remoção ordenada

```
int main() {
    abin a = ...;
    a = delete(3, a);
    return 0;
}
```



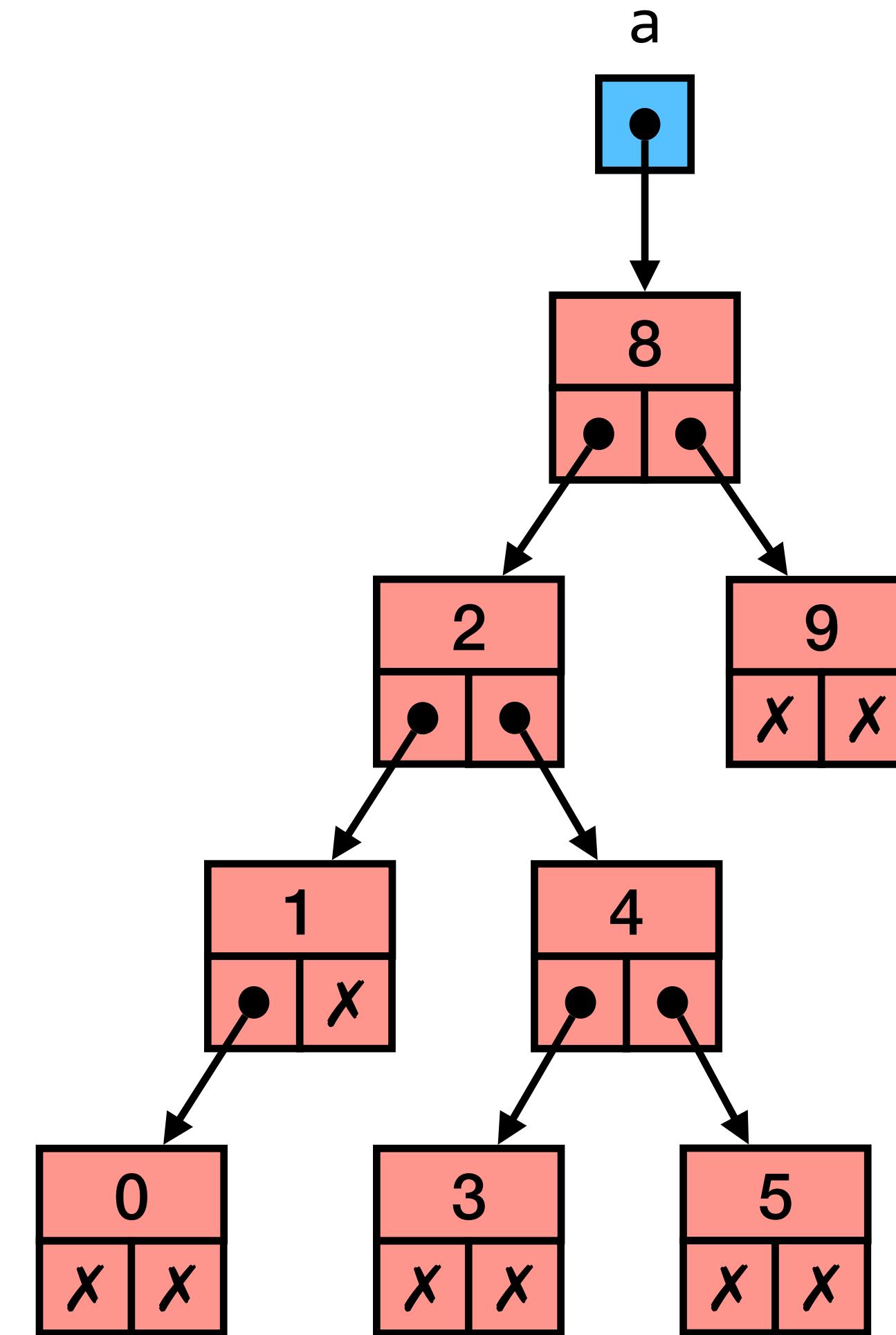
Remoção ordenada

```
int main() {
    abin a = ...;
    a = delete(3, a);
    return 0;
}
```



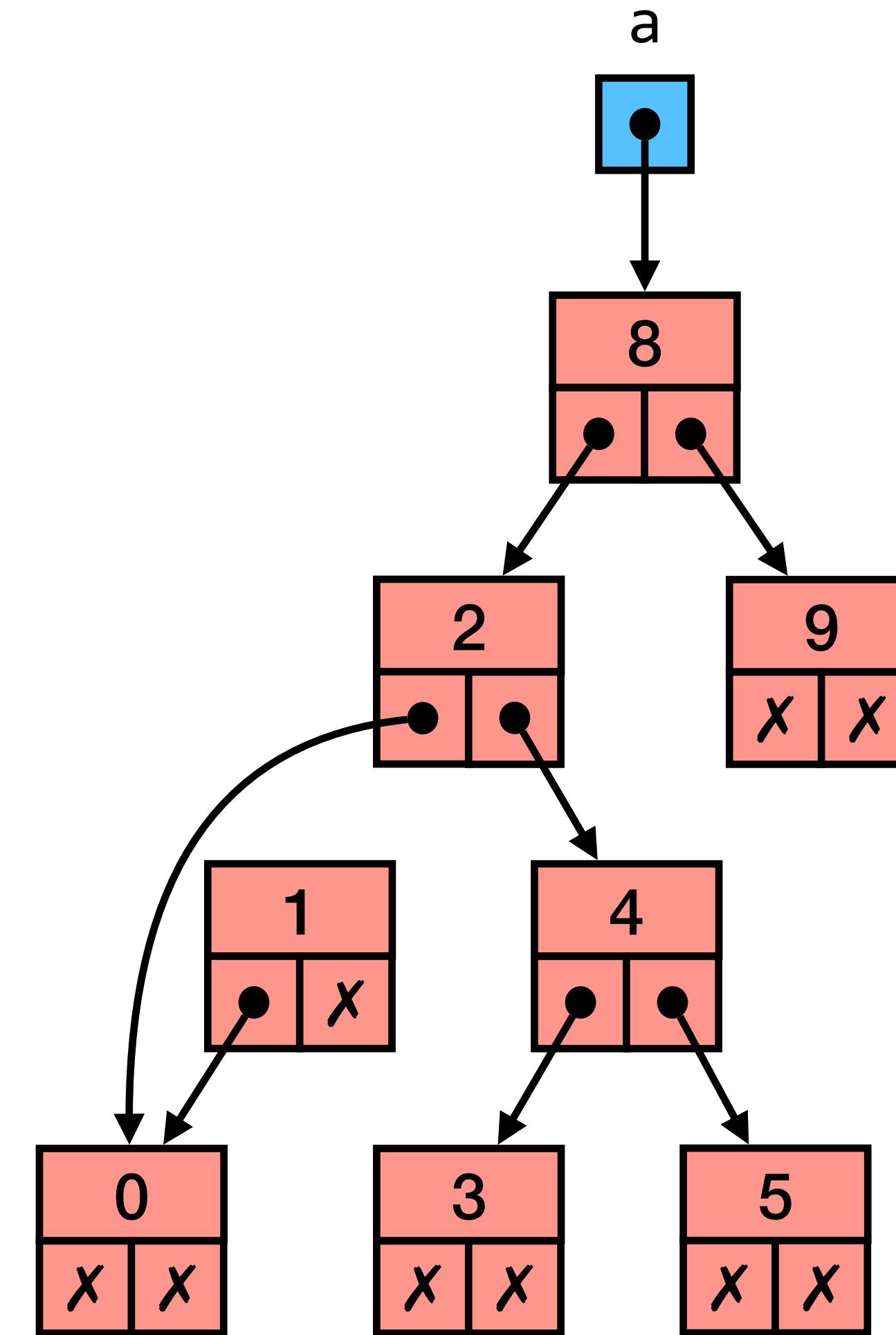
Remoção ordenada

```
int main() {
    abin a = ...;
    a = delete(1, a);
    return 0;
}
```



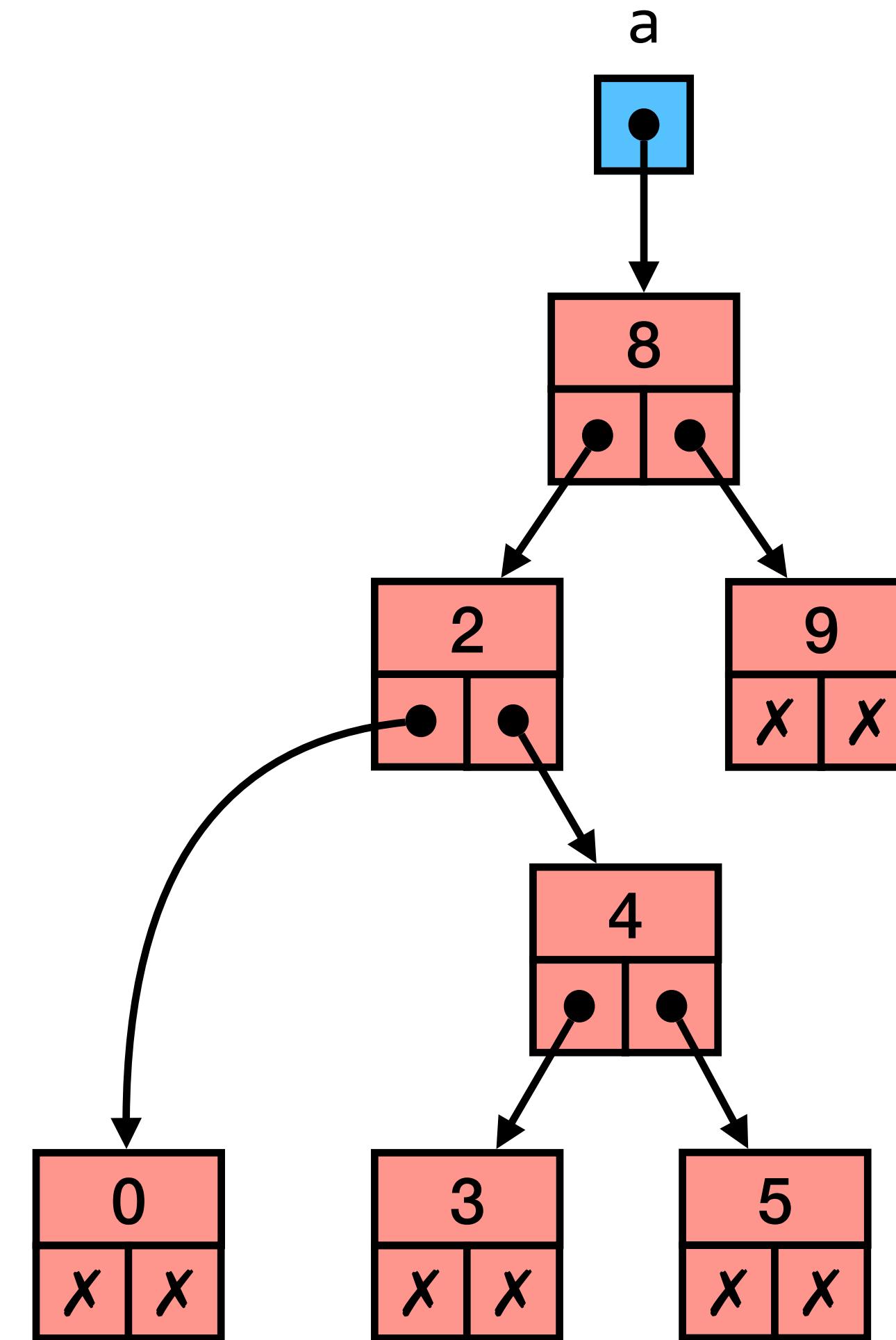
Remoção ordenada

```
int main() {
    abin a = ...;
    a = delete(1, a);
    return 0;
}
```



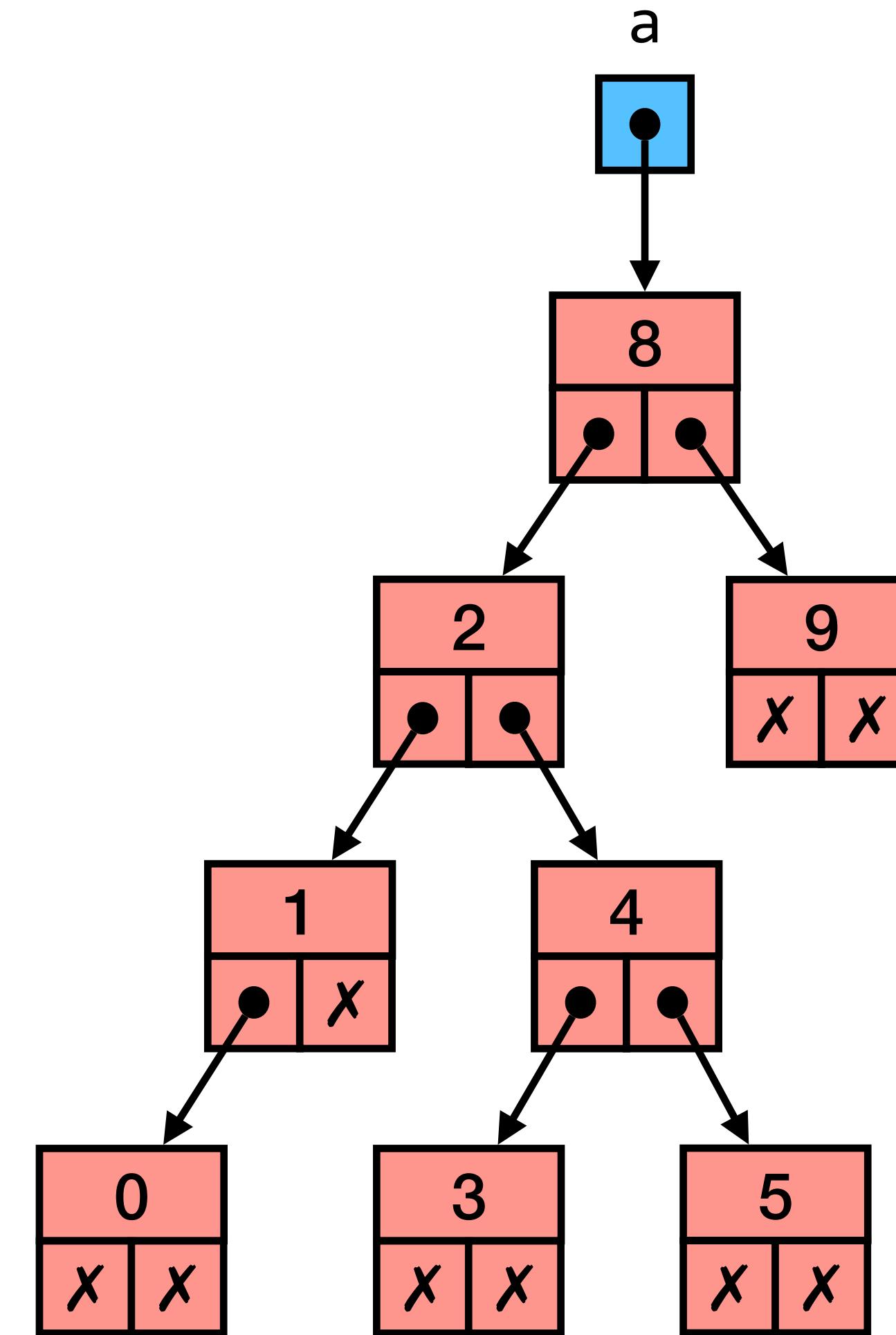
Remoção ordenada

```
int main() {
    abin a = ...;
    a = delete(1, a);
    return 0;
}
```



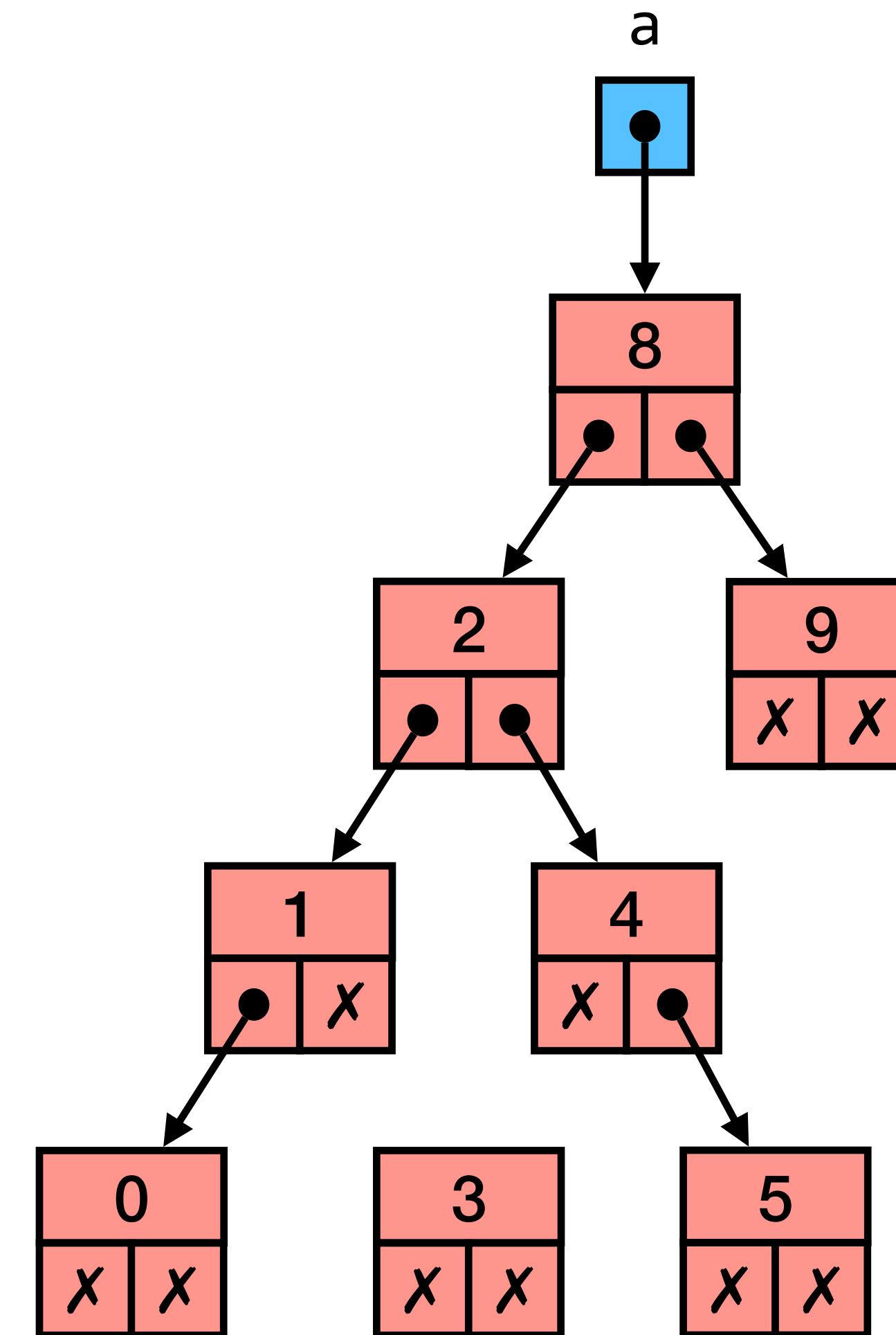
Remoção ordenada

```
int main() {
    abin a = ...;
    a = delete(2, a);
    return 0;
}
```



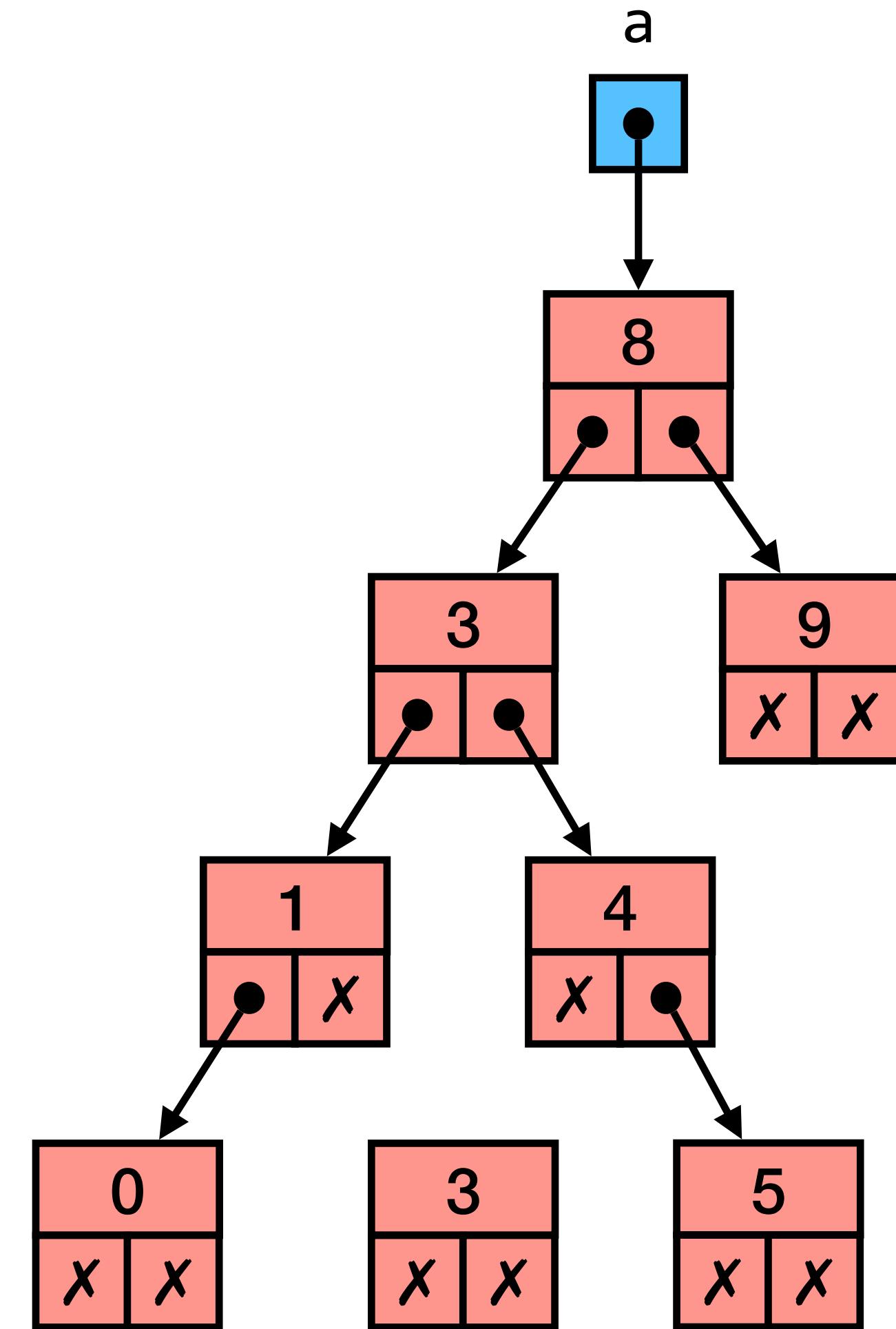
Remoção ordenada

```
int main() {
    abin a = ...;
    a = delete(2, a);
    return 0;
}
```



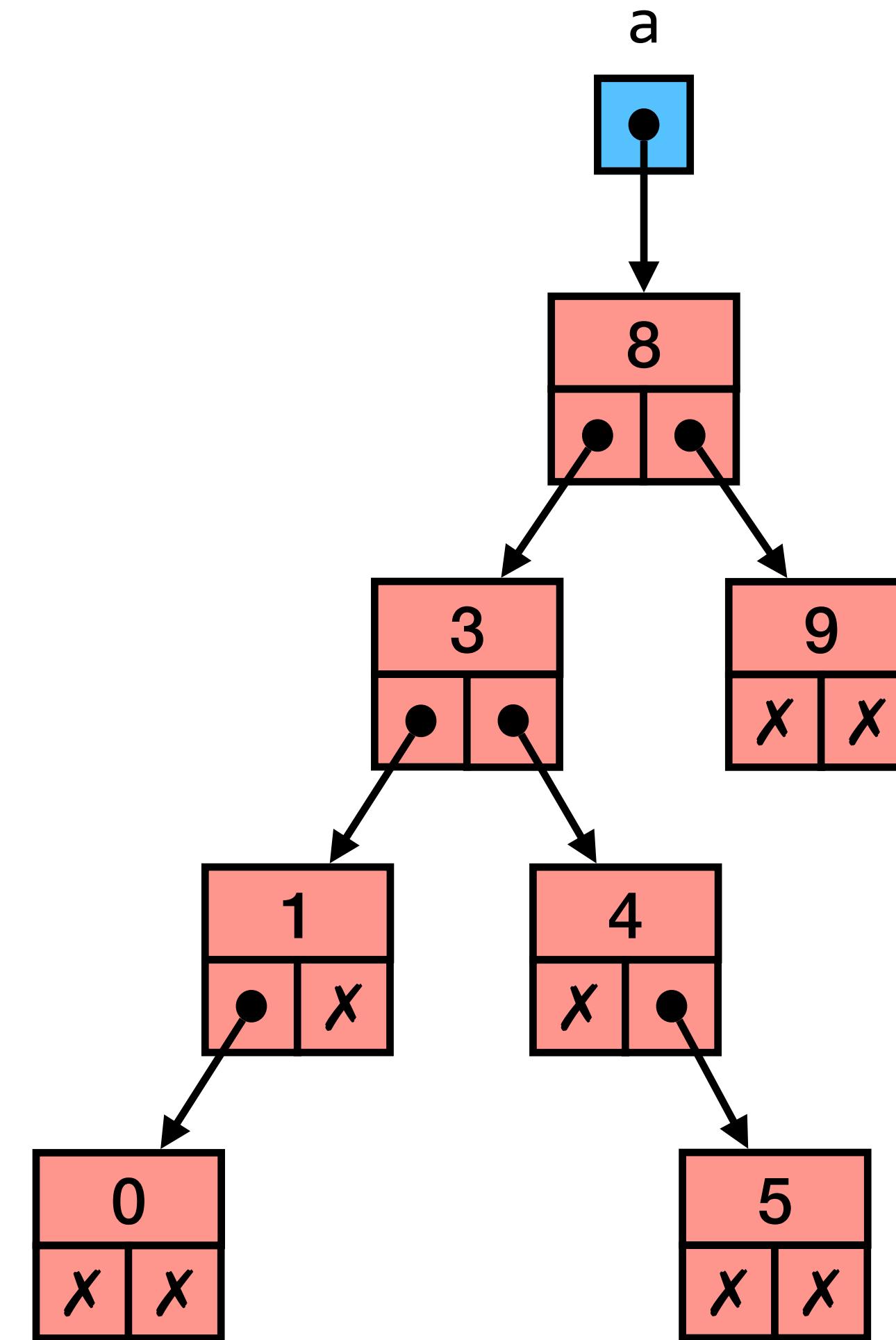
Remoção ordenada

```
int main() {
    abin a = ...;
    a = delete(2, a);
    return 0;
}
```



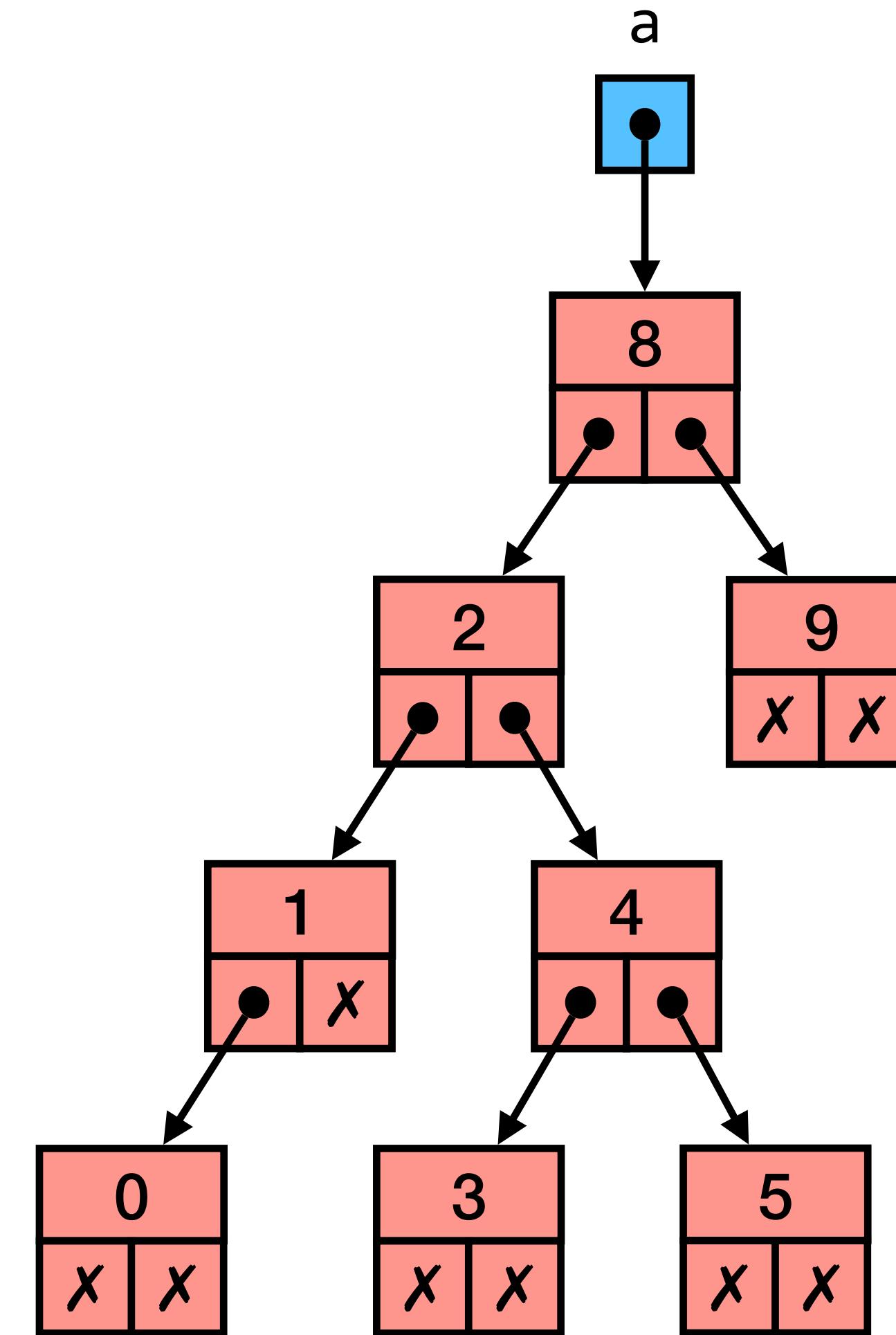
Remoção ordenada

```
int main() {
    abin a = ...;
    a = delete(2, a);
    return 0;
}
```



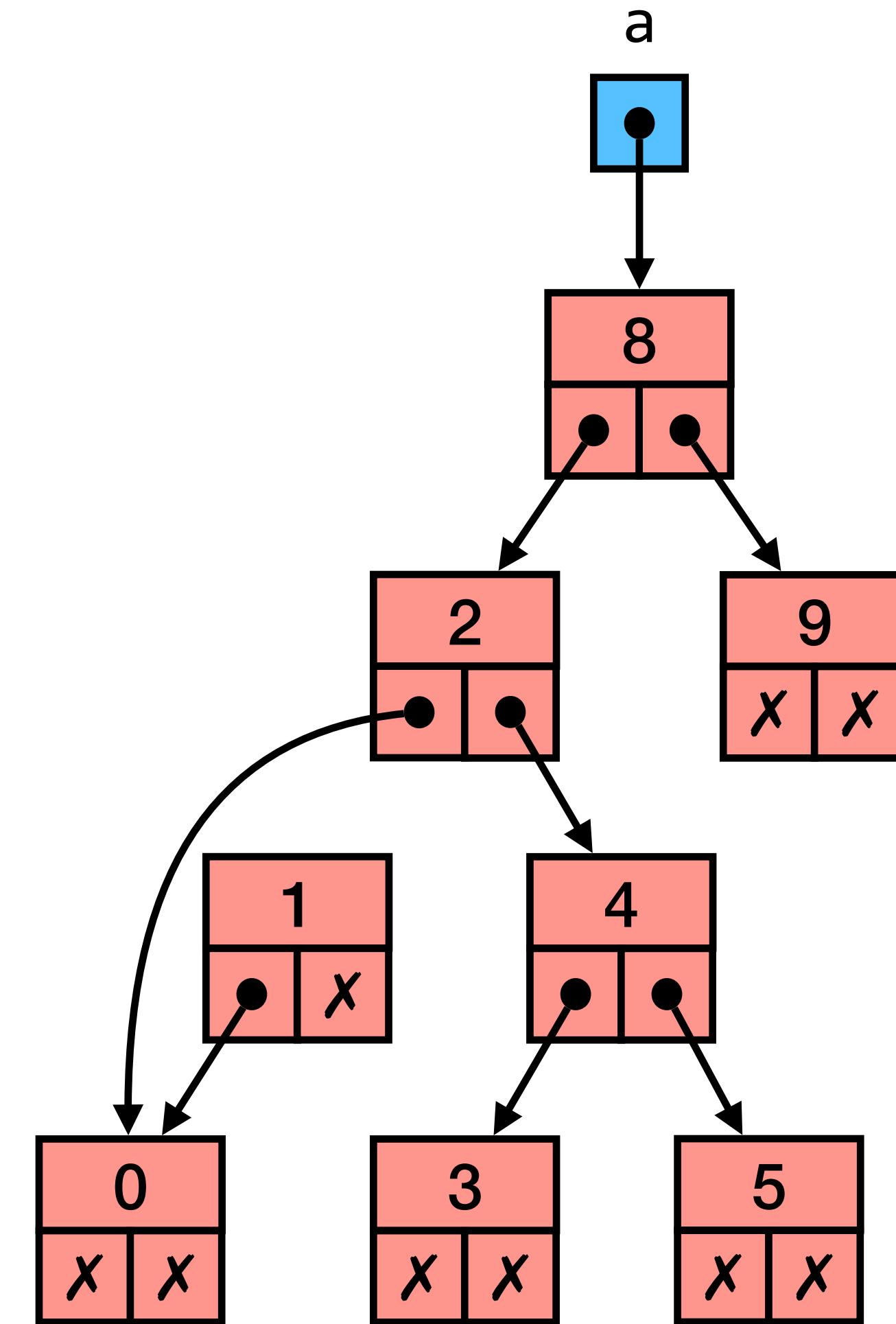
Remoção ordenada

```
int main() {
    abin a = ...;
    a = delete(2, a);
    return 0;
}
```



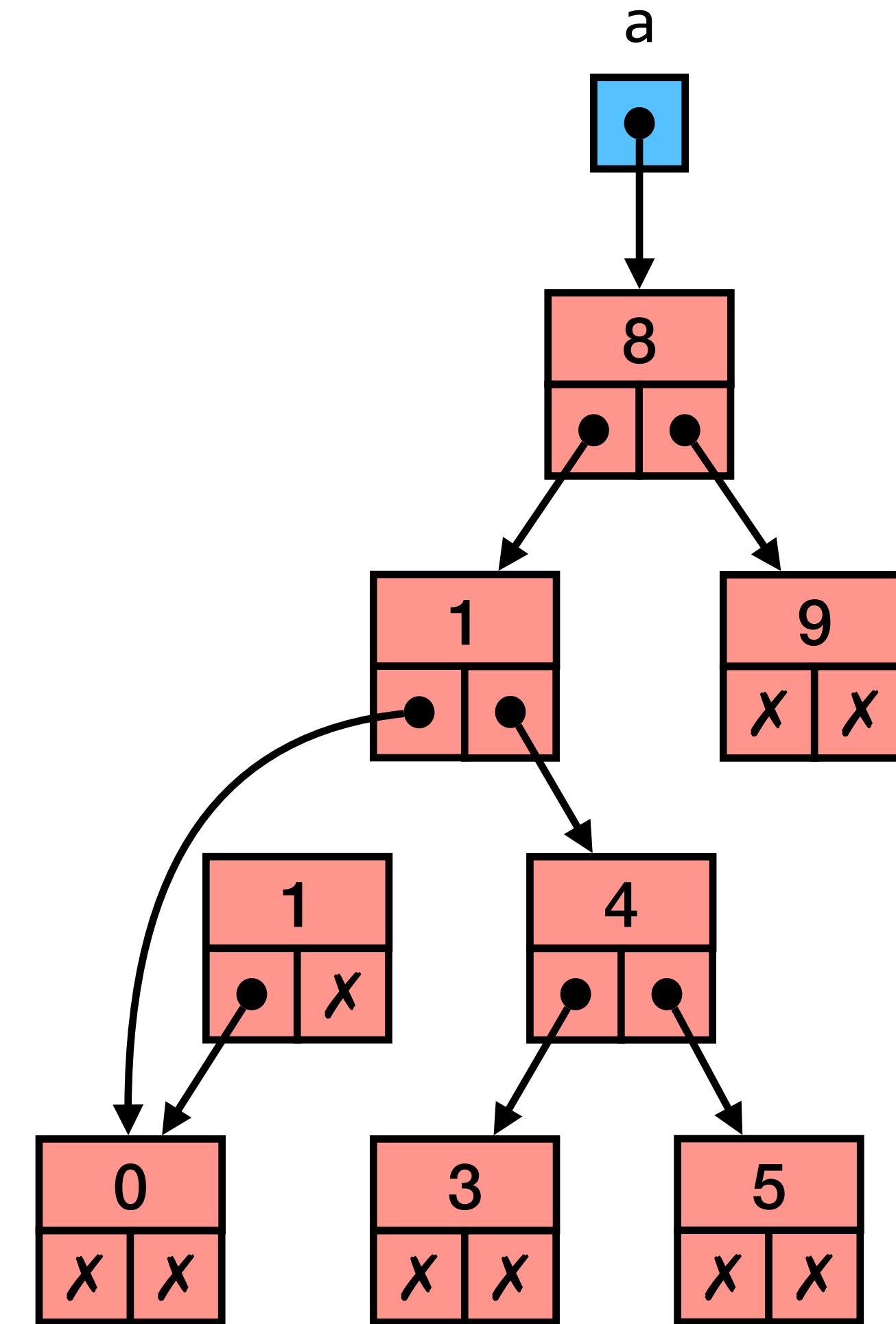
Remoção ordenada

```
int main() {
    abin a = ...;
    a = delete(2, a);
    return 0;
}
```



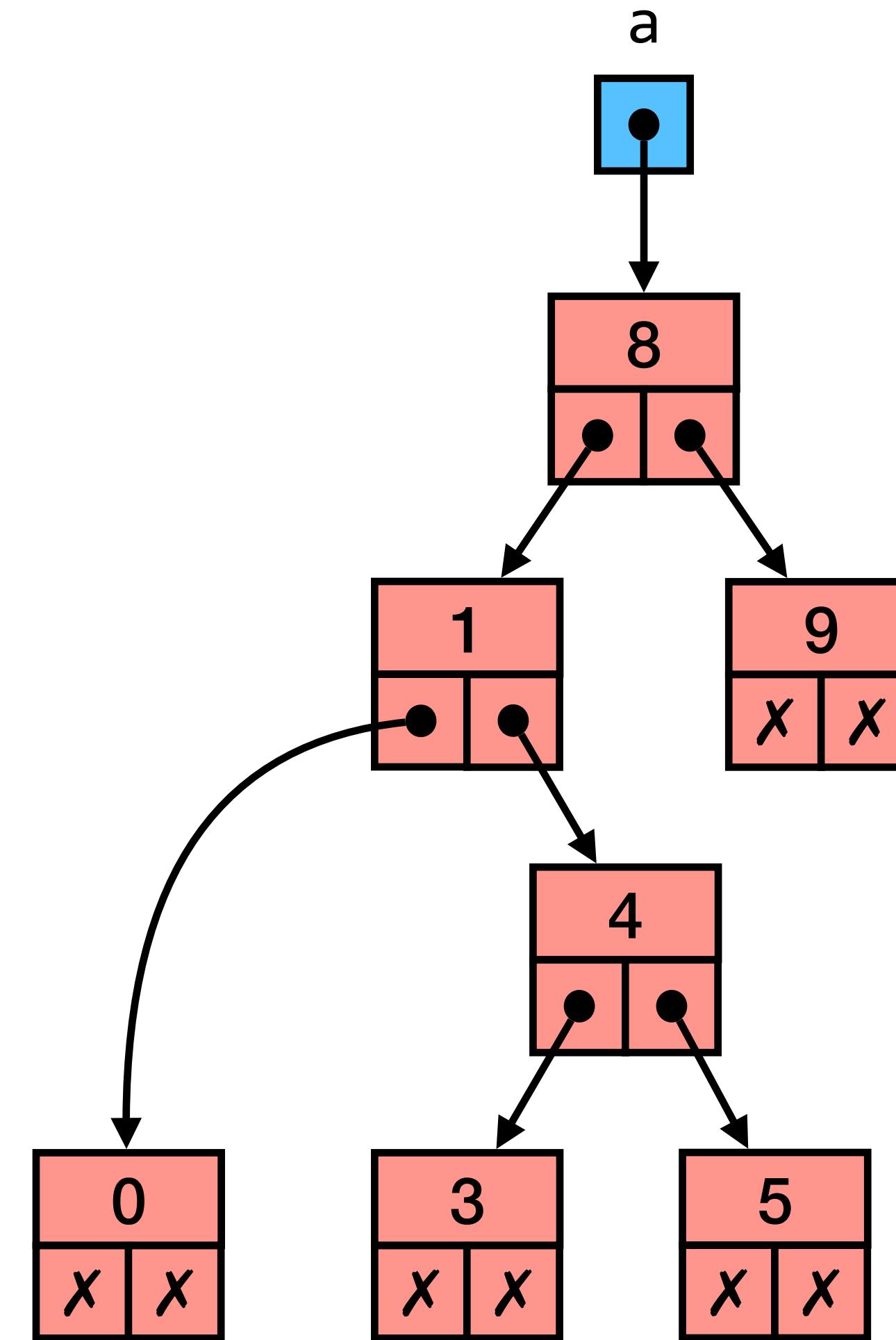
Remoção ordenada

```
int main() {
    abin a = ...;
    a = delete(2, a);
    return 0;
}
```

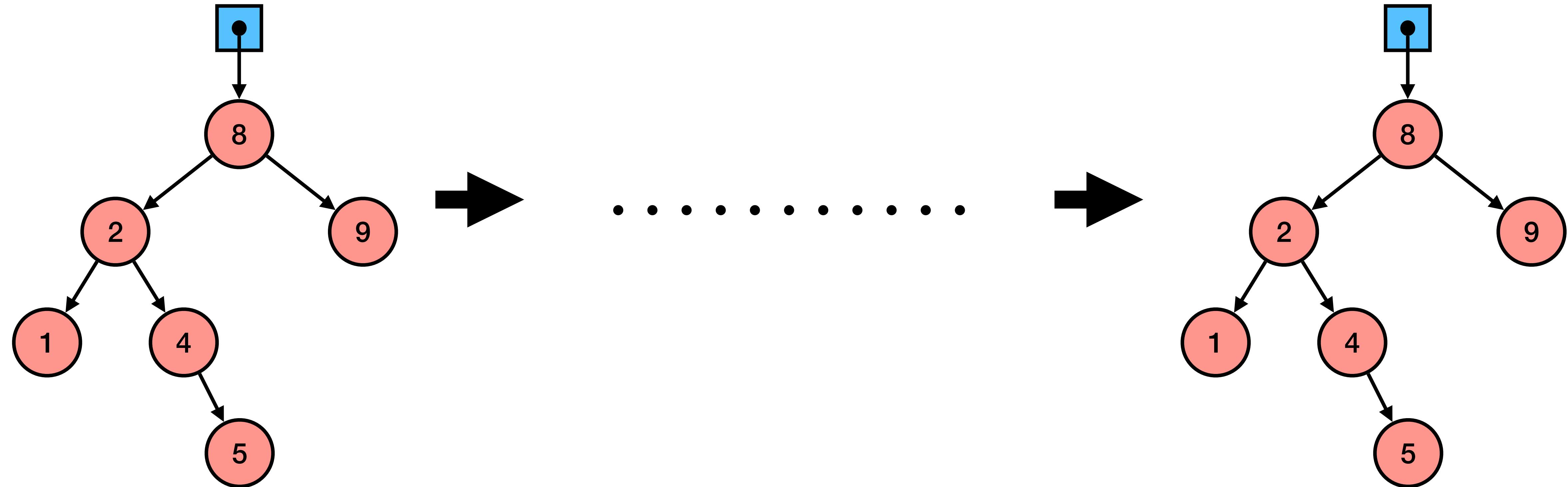


Remoção ordenada

```
int main() {
    abin a = ...;
    a = delete(2, a);
    return 0;
}
```



Serializar árvore de procura

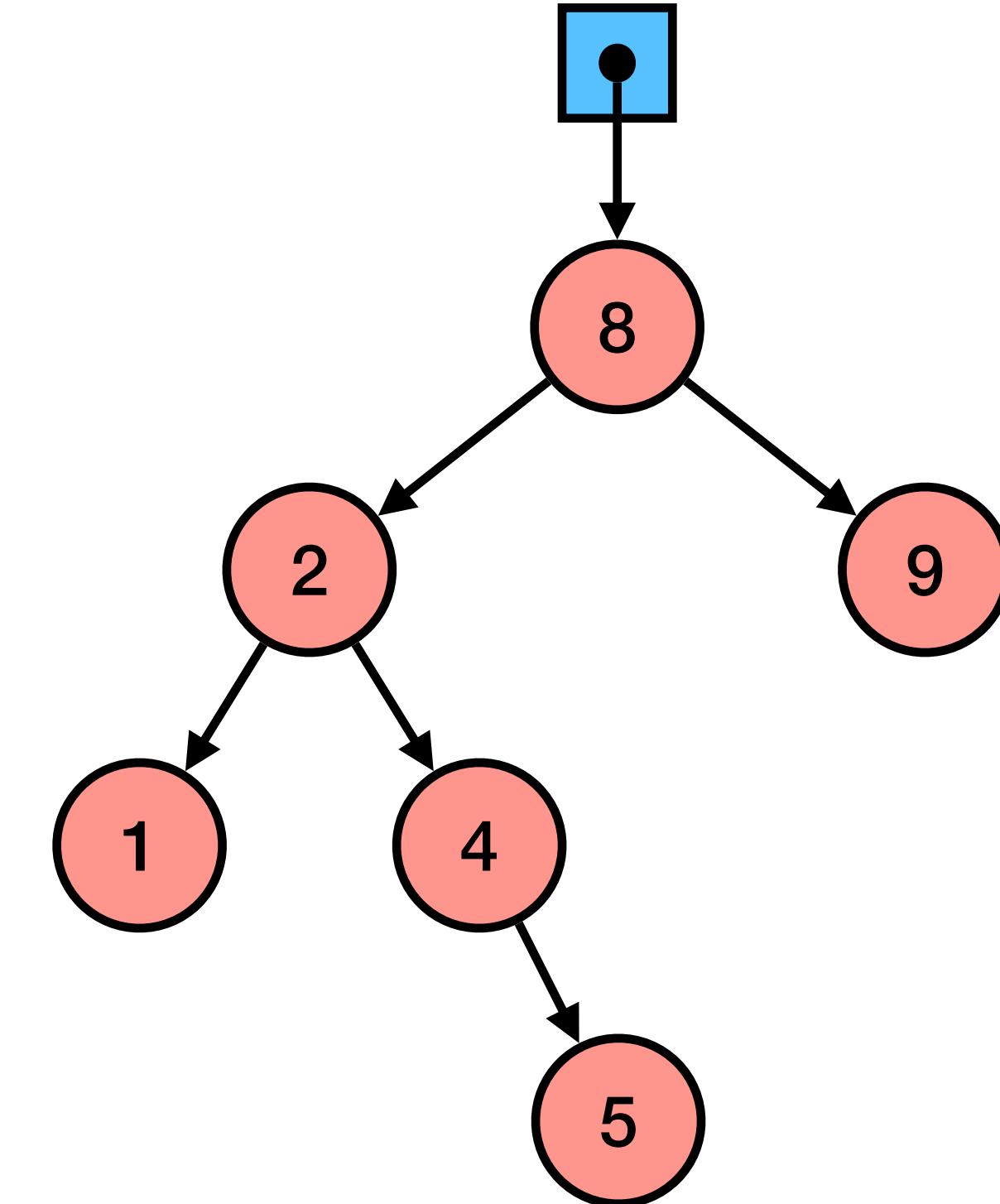


#31 Que travessias podem ser invertidas?

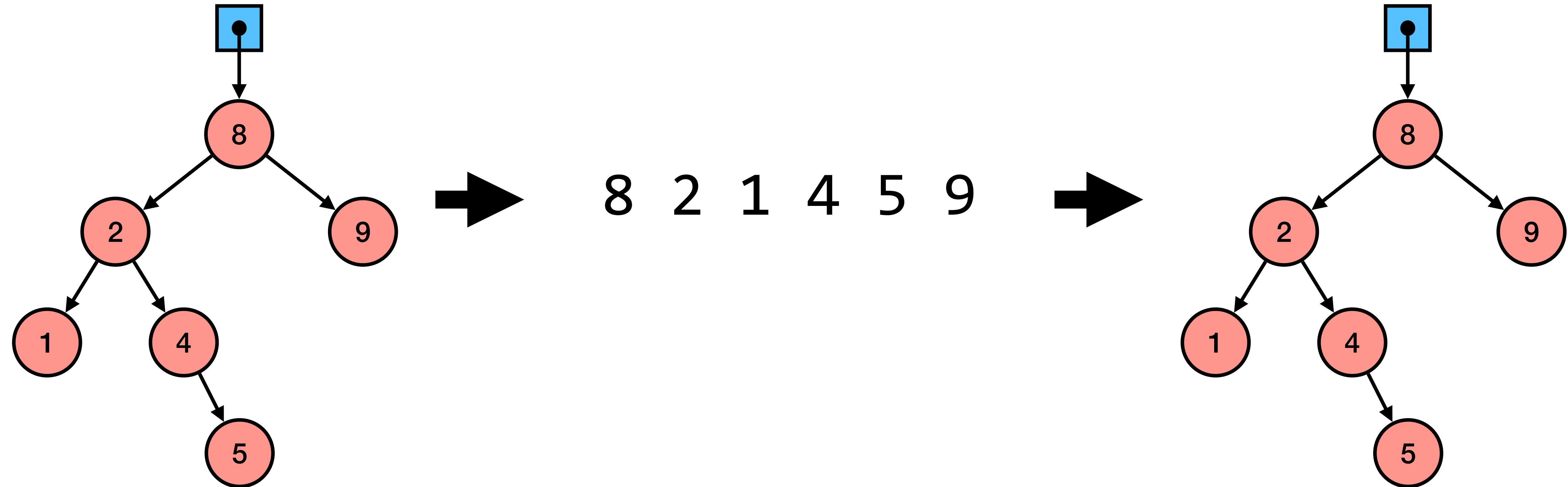
```
int toArrayInorder(abin a, int v[]);
```

```
int toArrayPreorder(abin a, int v[]);
```

```
int toArrayPostorder(abin a, int v[]);
```



Serializar árvore de procura

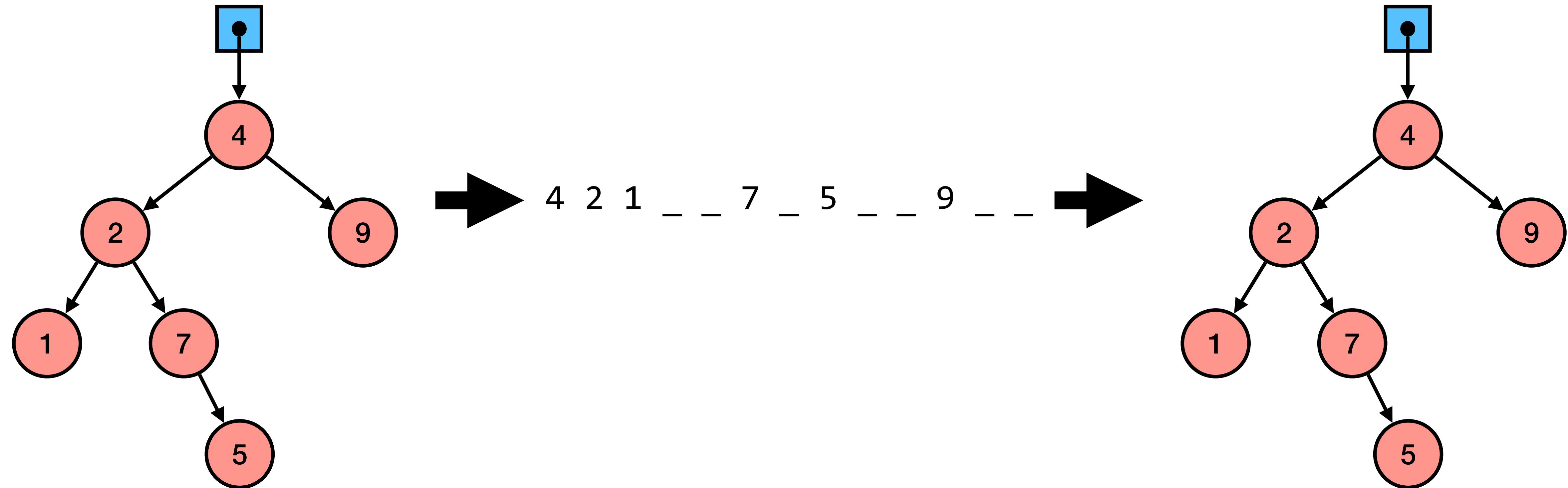


Inverter array preorder

```
int aux(abin *a, int *max, int v[], int N) {
    abin l, r; int nl, nr;
    if (N == 0 || (max != NULL && v[0] > *max)) {
        *a = NULL;
        return 0;
    }
    nl = aux(&l, v, v+1, N-1);
    nr = aux(&r, NULL, v+nl+1, N-nl-1);
    *a = mkroot(v[0], l, r);
    return 1+nl+nr;
}
```

```
abin fromArrayPreorder(int v[], int N) {
    abin a;
    aux(&a,NULL,v,N);
    return a;
}
```

Serializar árvore arbitrária



Aula 14

#32 Qual a palavra mais frequente dos Lusíadas?

- as armas
- e
- os barões assinalados
- que
- da ocidental praia lusitana



#33 Qual a melhor forma de implementar o histograma?

- A. Lista desordenada, novas palavras inseridas no início
- B. Lista desordenada, novas palavras inseridas no fim
- C. Lista ordenada por palavra
- D. Árvore binária de procura ordenada por palavra



