

# Variáveis

# Conceitos

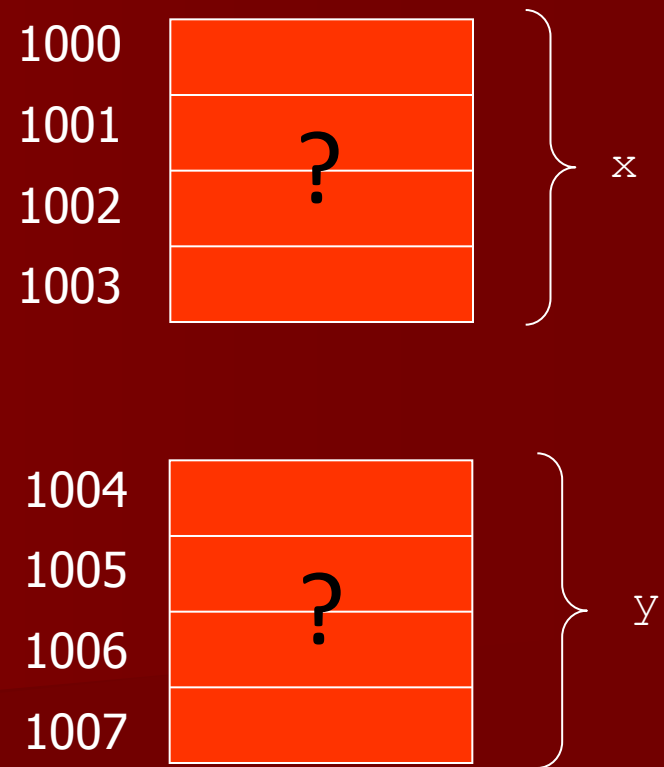
- São alocadas em memória;
- Elas ocupam uma quantidade de bytes que depende do tipo da variável;
- Esses bytes são alocados de forma contígua na memória;
- O **valor** de uma variável corresponde ao dado que está armazenado nesses bytes;
- O **endereço** de uma variável corresponde ao endereço de memória onde está alocado o primeiro byte que conjunto de bytes que a compõe;
- O endereço de uma variável é determinado pelo sistema de execução.

# Linguagem C

- Seja “x” o nome de uma variável inteira (“int”);
- “x” ocupa 4 bytes consecutivos de memória;
- Quando usado em expressões, “x” denota o valor inteiro armazenado nesses 4 bytes;
- Quando usado no lado esquerdo de um comando de atribuição, “x” denota o endereço da variável “x”;
- “&x” denota o endereço da variável “x” e corresponde ao endereço de memória do primeiro byte dos 4 que compõem “x”;
- “&” é uma operação unária que retorna o endereço da variável fornecida como argumento.

```
int x,y;
```

- “x” e “y” ocupam 4 bytes consecutivos cada;
- “x” (“y”) inicia no endereço 1000 (1004) e termina no endereço 1003 (1007);
- O endereço de “x” (“y”) é 1000 (1004);
- Os valores iniciais de “x” e de “y” são indeterminados;



**x=0 ;**

**y=x+1 ;**

**x=&y+2 ;**

x

x

x

1000  
1001  
1002  
1003

0

1000  
1001  
1002  
1003

0

1000  
1001  
1002  
1003

1006

y

y

y

1004  
1005  
1006  
1007

?

1004  
1005  
1006  
1007

1

1004  
1005  
1006  
1007

1

**Ponteiros**

# Conceitos

- É uma variável que contém o endereço de outra variável;
- O valor armazenado num ponteiro corresponde ao endereço de outra variável, mesmo que esta outra também seja um ponteiro;
- Deve-se “dereferenciar” a variável ponteiro para acessar o valor a que ele se refere;
- Variáveis do tipo ponteiro permitem múltiplas formas de acesso a um mesmo valor;

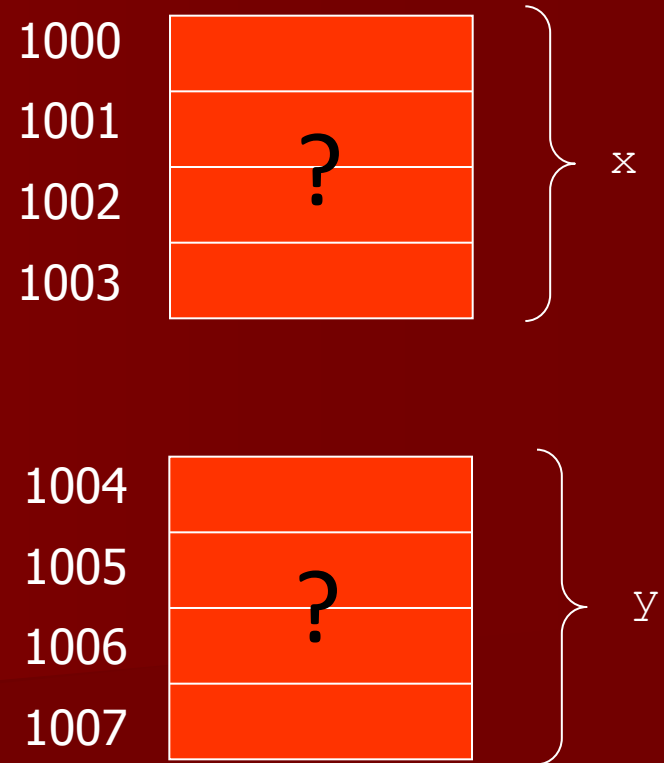
# Linguagem C

- Deve-se definir o “tipo” do valor que será armazenado na posição de memória indicada pelo endereço contido na variável ponteiro (tipo do valor apontado);
- Uma variável do tipo ponteiro ocupa 4 bytes de memória, independentemente do tipo do valor apontado;
- Para acessar o valor apontado pela variável ponteiro deve-se usar o operador “\*”;
- “\*” é uma operação unária que retorna o valor armazenado no endereço memória contido na variável usada como argumento da operação.



```
int x, *y;
```

- “x” e “y” ocupam 4 bytes consecutivos cada;
- “x” (“y”) inicia no endereço 1000 (1004) e termina no endereço 1003 (1007);
- O endereço de “x” (“y”) é 1000 (1004);
- Os valores iniciais de “x” e de “y” são indeterminados;
- “x” é do tipo inteiro;
- “y” é do tipo ponteiro para inteiro;



`y=&x;`

`x=1;`

`*y=*y+2;`

x

x

x

1000	
1001	
1002	?
1003	

1000	
1001	
1002	1
1003	

1000	
1001	
1002	3
1003	

y

y

y

1004	
1005	
1006	1000
1007	

1004	
1005	
1006	1000
1007	

1004	
1005	
1006	1000
1007	

`x=x+*y;`

x

1000	
1001	
1002	6
1003	

`y=y+1;`

x

1000	
1001	
1002	6
1003	

`*y=0;`

x

1000	
1001	
1002	6
1003	

y

1004	
1005	1000
1006	
1007	

y

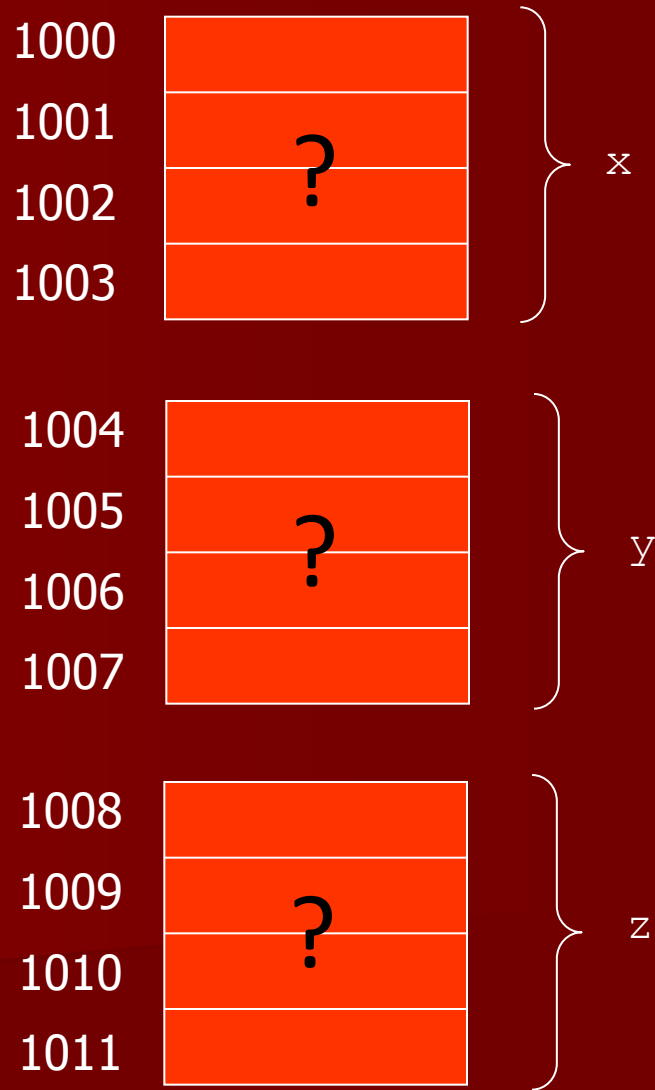
1004	
1005	1004
1006	
1007	

y

1004	
1005	
1006	0
1007	

```
int x,*y,*z;
```

- “x” é do tipo inteiro;
- “y” é do tipo inteiro;
- “z” é do tipo ponteiro para inteiro;



**x=0 ;**

x

1000	
1001	
1002	0
1003	

y

1004	
1005	?
1006	
1007	

z

1008	
1009	?
1010	
1011	

**y=&x ;**

x

1000	
1001	
1002	0
1003	

y

1004	
1005	1000
1006	
1007	

z

1008	
1009	?
1010	
1011	

**z=&x ;**

x

1000	
1001	
1002	0
1003	

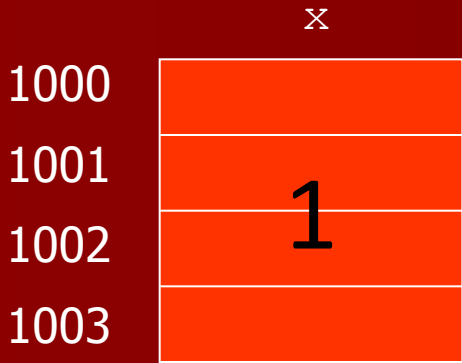
y

1004	
1005	1000
1006	
1007	

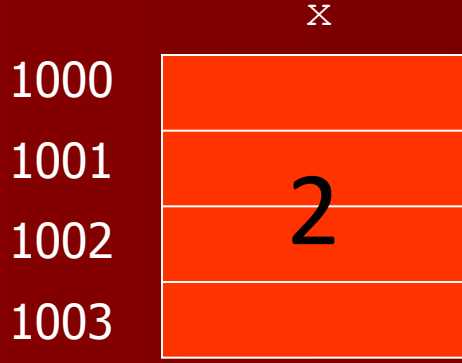
z

1008	
1009	1000
1010	
1011	

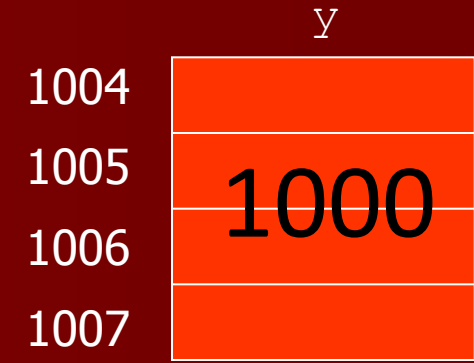
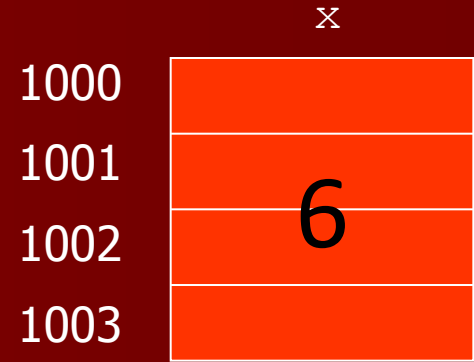
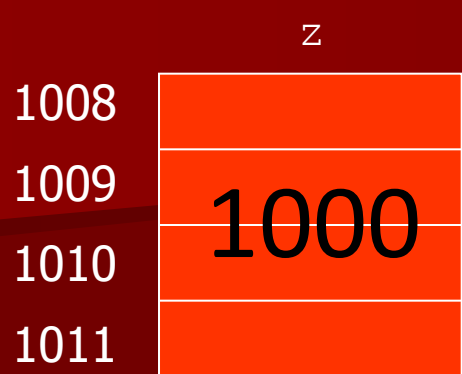
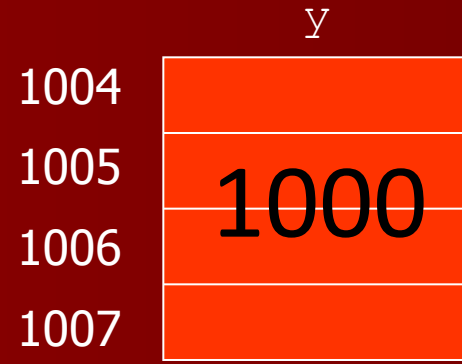
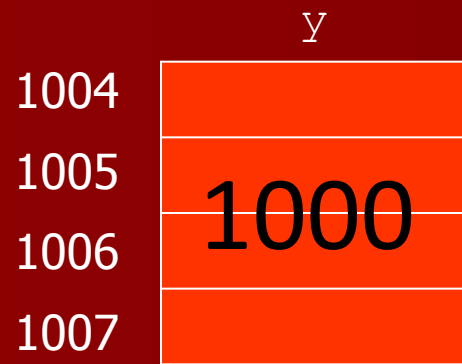
`*y=*y+1;`



`*z=*z+1;`

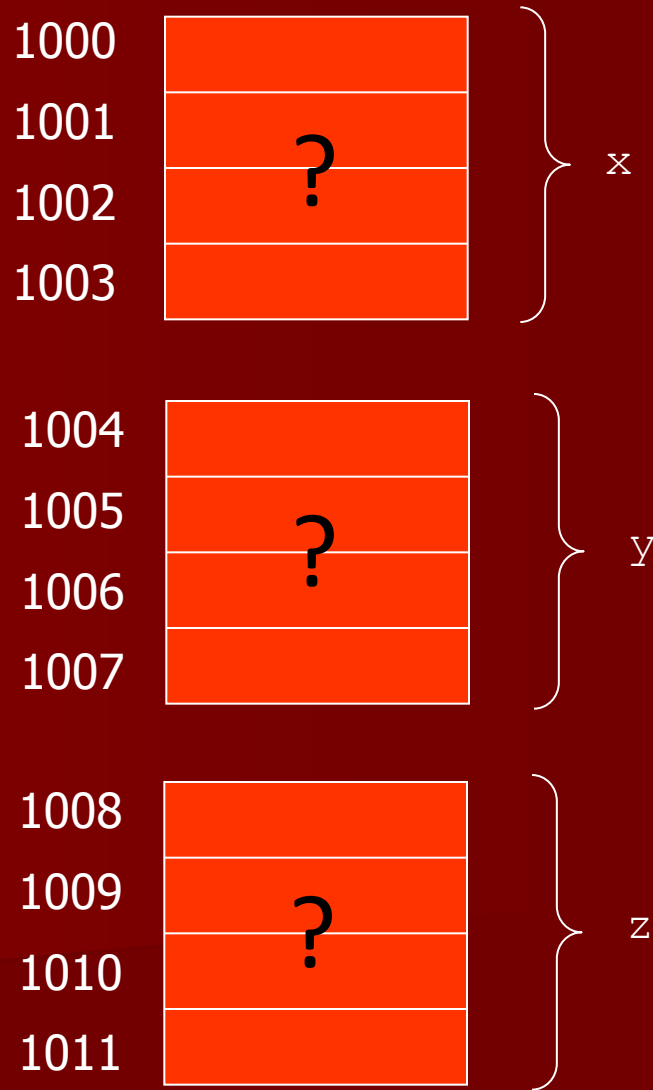


`*y=*y+*z+x;`



```
int x, *y, **z;
```

- “x” é do tipo inteiro;
- “y” é do tipo ponteiro para inteiro;
- “z” é do tipo ponteiro para ponteiro para inteiro;



**x=0 ;**

x

1000	
1001	0
1002	
1003	

**y=&x ;**

x

1000	
1001	0
1002	
1003	

**z=&y ;**

x

1000	
1001	0
1002	
1003	

y

1004	
1005	?
1006	
1007	

y

1004	
1005	1000
1006	
1007	

y

1004	
1005	1000
1006	
1007	

z

1008	
1009	?
1010	
1011	

z

1008	
1009	?
1010	
1011	

z

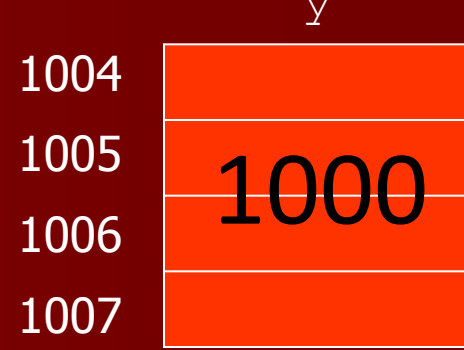
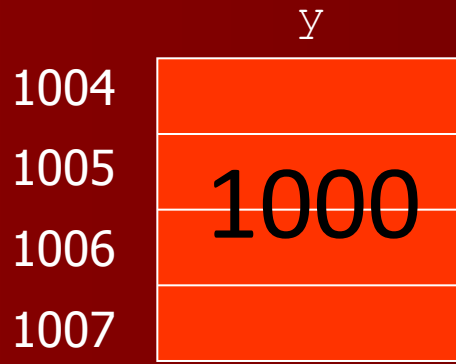
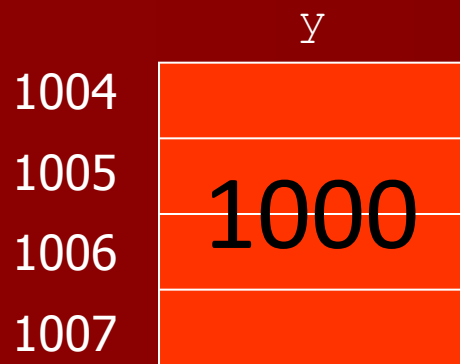
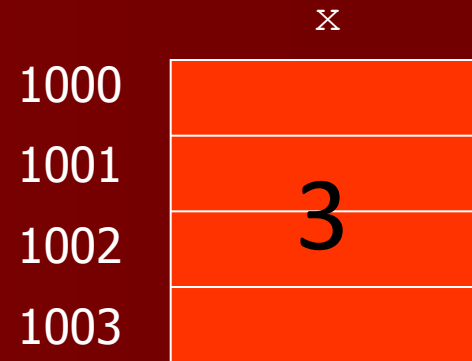
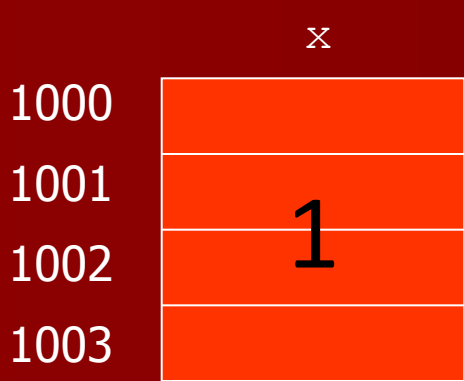
1008	
1009	1004
1010	
1011	



**\*\*z=\*\*z+1;**

**\*y=\*y+1;**

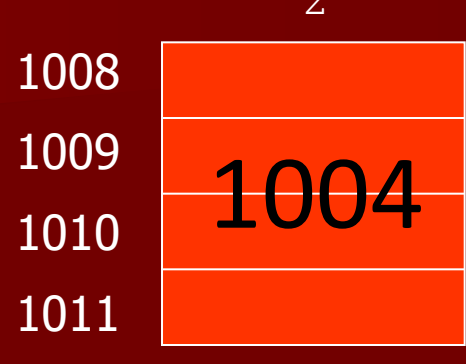
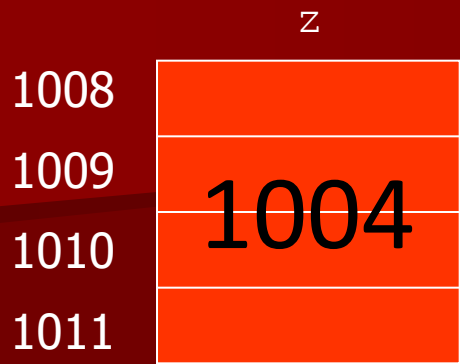
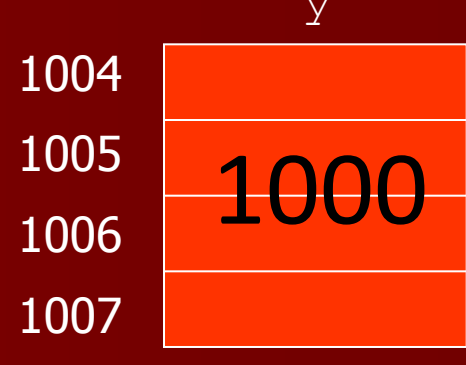
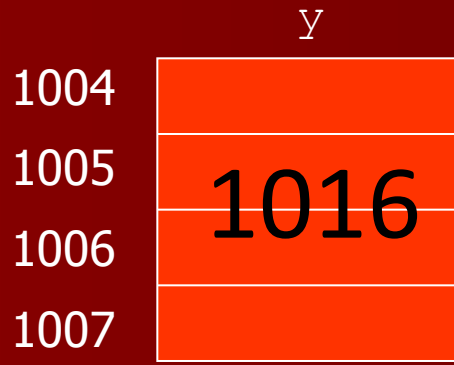
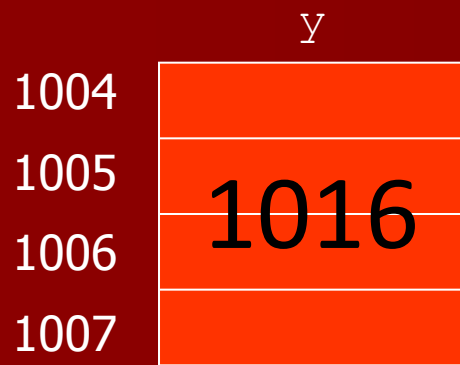
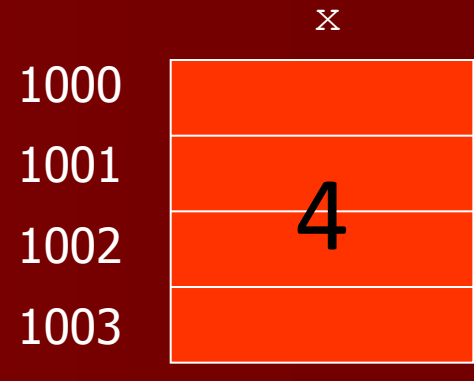
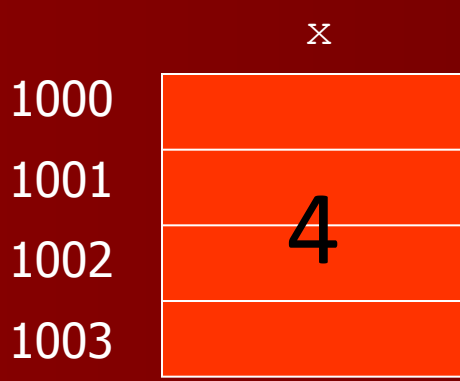
**x=x+1;**



`*z=&x+4 ;`

`*(y-4)=*(y-4)+1 ;`

`y=y-4 ;`



# Vetores

# Conceitos

- Um vetor é um agregado de dados homogêneo, pois ele é composto de uma coleção de valores de um mesmo tipo;
- Normalmente é usado para representar coleções de valores afins;
- Os valores armazenados nos elementos de um vetor podem ser consultados e modificados de forma independente uns dos outros;
- Em casos especiais o vetor pode ser tratado como um objeto único.

# Conceitos

- Os elementos de um vetor são alocados de forma contígua na memória do computador;
- O primeiro elemento é referenciado pelo índice inicial e o último pelo índice final;
- Índice não elemento  $\neq$  endereço do elemento;
- A operação de indexação permite selecionar um elemento do agregado;
- A indexação é normalmente feita através de expressões inteiras;
- A indexação com valores fora dos limites especificados constitui erro, o qual muitas vezes é detectado apenas em tempo de execução e pode provocar situações imprevisíveis.

# Conceitos

Atributos de um vetor:

- Nome;
- Tipo dos seus elementos;
- Quantidade de elementos;
- Índice do primeiro elemento;
- Índice do último elemento.

# Linguagem C

- Deve-se especificar a quantidade de elementos do vetor;
- A expressão que define a quantidade de elementos do vetor é avaliada em tempo de execução;
- O índice inicial é sempre 0 (zero);
- O índice final é a quantidade de elementos menos 1 (um);
- A indexação de um vetor é feita através do operador “[...]” logo após o nome do vetor;

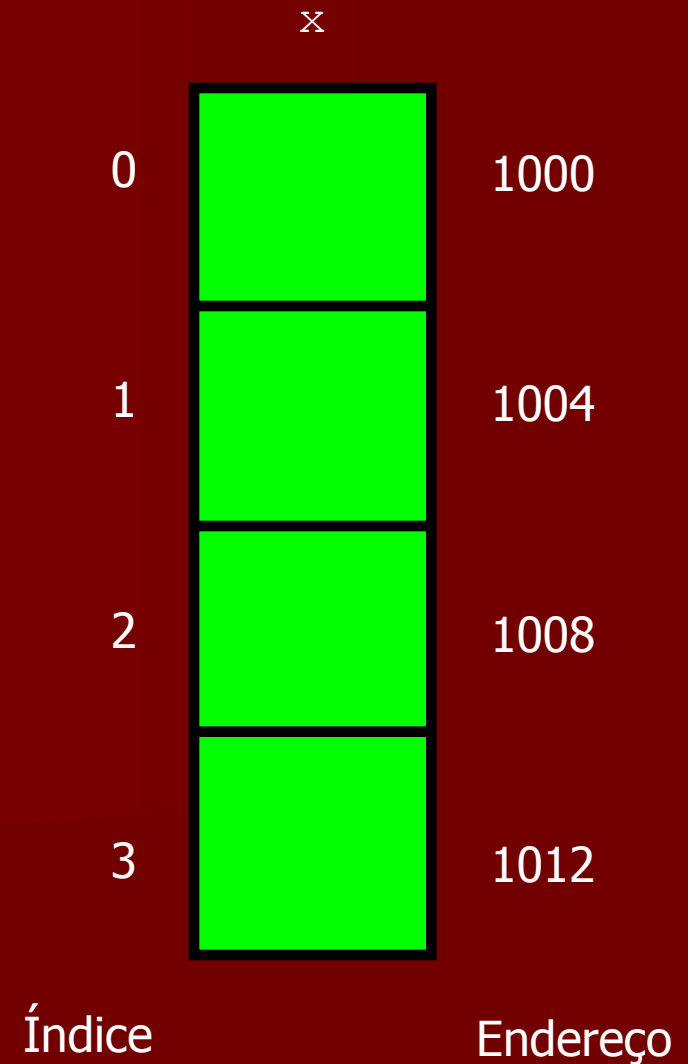
# Linguagem C

- A expressão indexadora, utilizada dentro dos colchetes, é avaliada em tempo de execução e deve resultar num valor do tipo inteiro (ou “char”);
- A declaração de um vetor implica a alocação em memória de um objeto com tamanho total correspondente ao número de elementos multiplicado pelo tamanho do elemento.

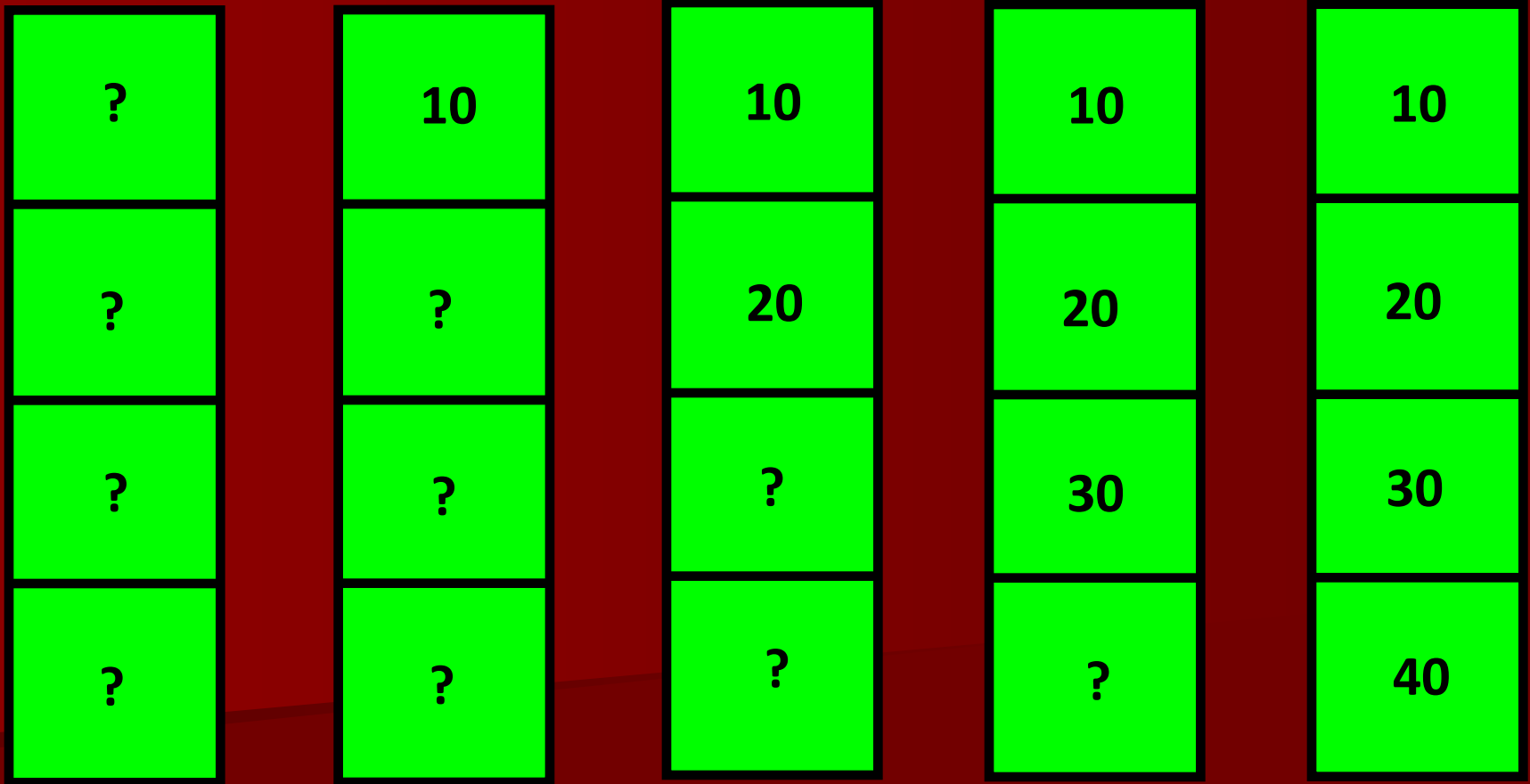


```
int x[4];
```

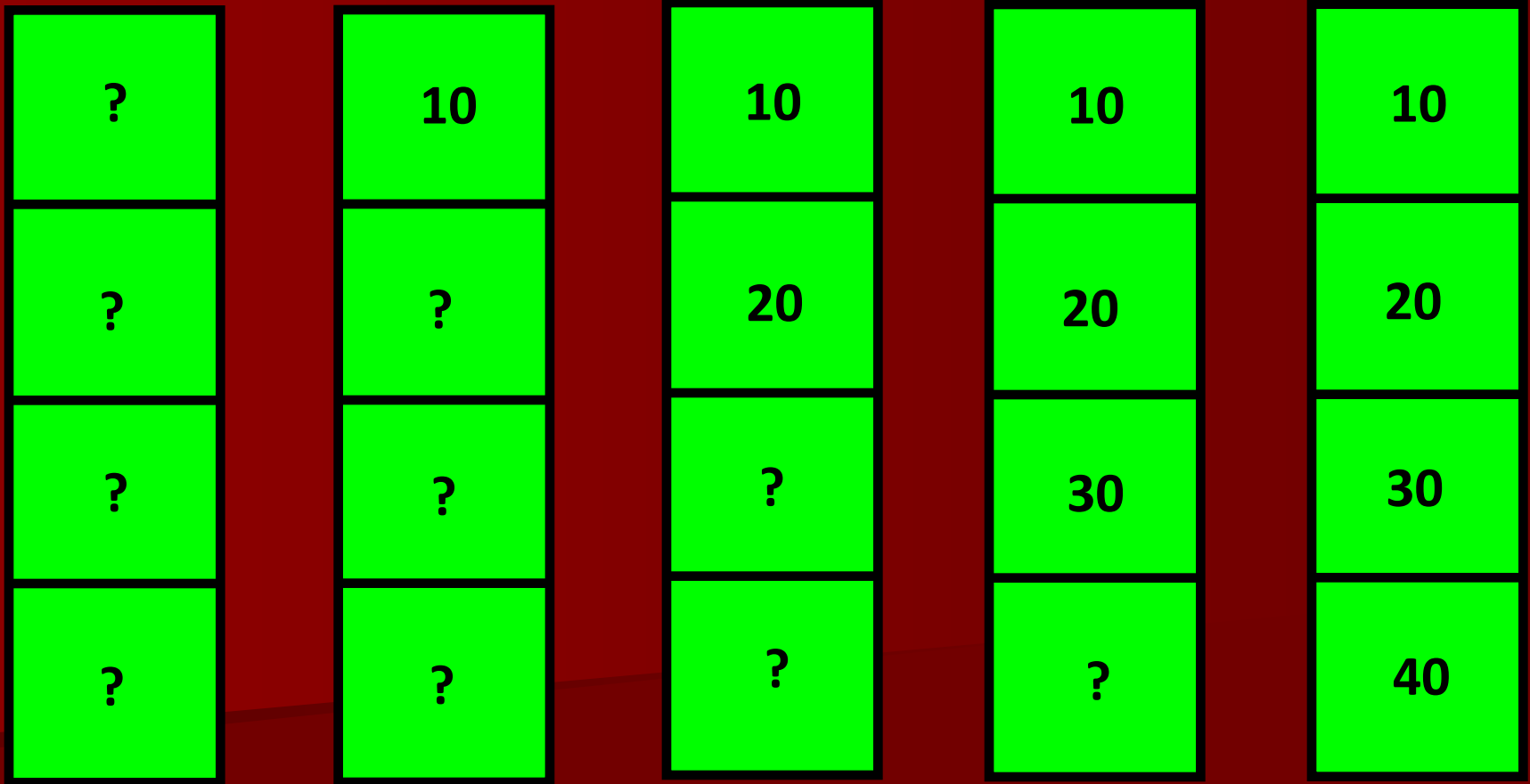
- Nome do vetor: “x”;
- Tipos dos seus elementos: inteiro;
- Quantidade de elementos: 4;
- Índice do primeiro elemento: 0;
- Índice do último elemento: 3;
- Os elementos 0, 1, 2 e 3 estão alocados, respectivamente, nos endereços de memória 1000, 1004, 1008 e 1012;
- O vetor ocupa um total de 16 bytes, iniciando no endereço 1000 e terminando no endereço 1015



```
x[0]=10; x[1]=20; x[2]=30; x[3]=40;
```



```
for (i=0;i<4;i++) x[i]=(i+1)*10;
```



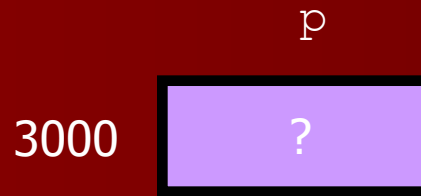
**Vetores**

**x**

**ponteiros**

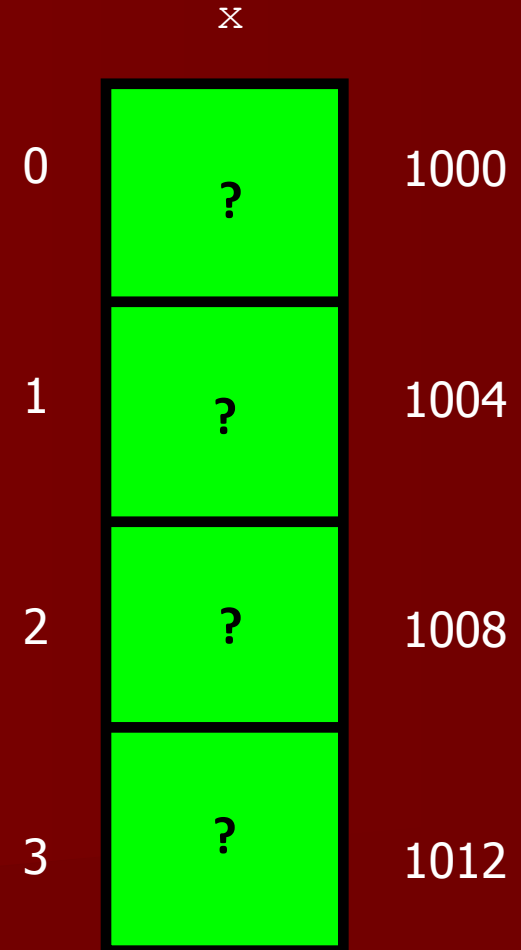
# Linguagem C

- Nomes de vetores podem ser manipulados como ponteiros;
- Nomes de vetores podem ser atribuídos para variáveis do tipo ponteiro;
- Variáveis do tipo ponteiro podem ser usadas para acessar os elementos de um vetor;
- Operações aritméticas sobre valores do tipo ponteiro (soma e subtração apenas) consideram o tamanho do objeto apontado e fazem os ajustes correspondentes automaticamente.



```
int x[4], *p;
```

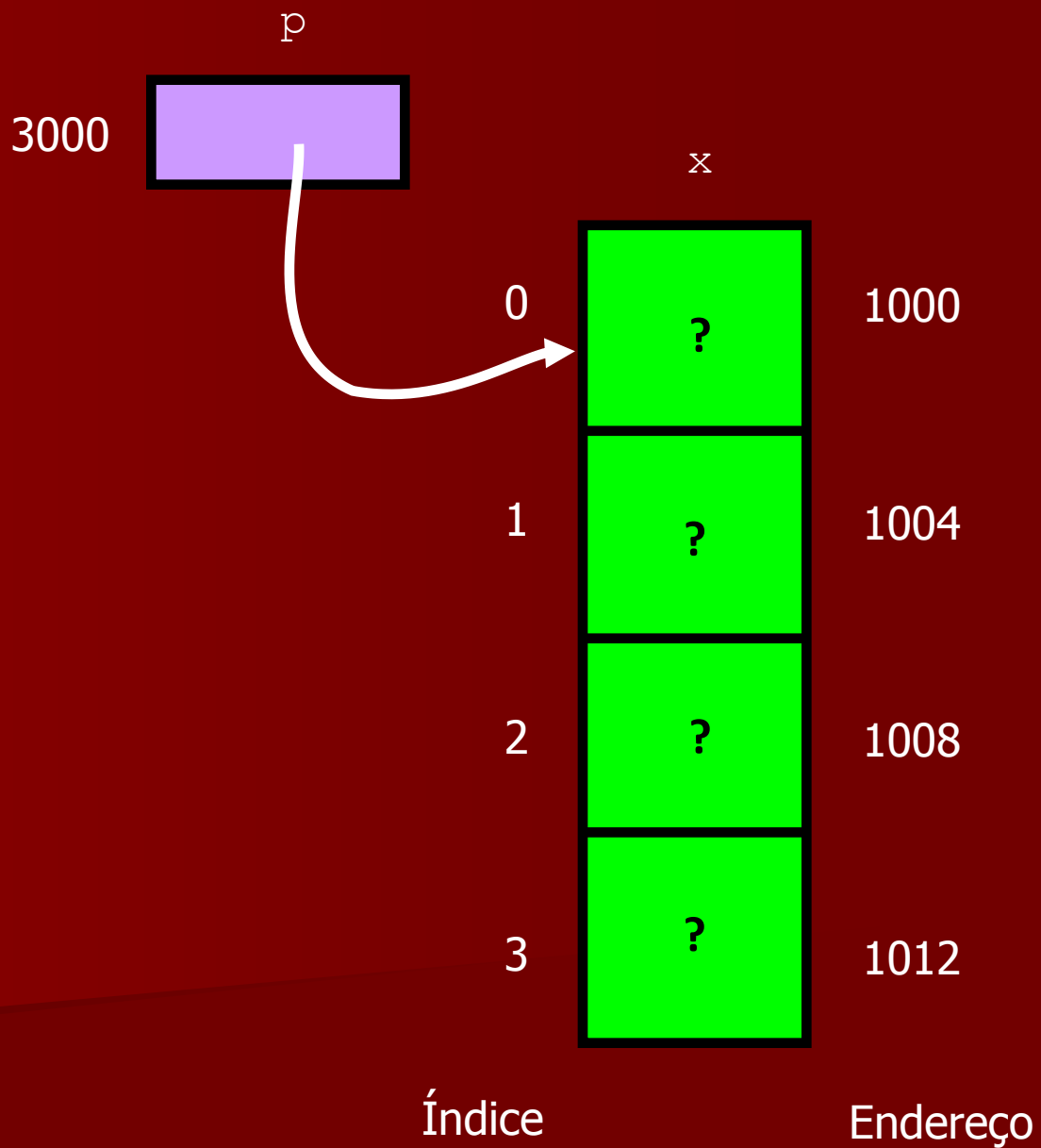
- “p” é um ponteiro para inteiro;
- Como “x” é um vetor de inteiros, então “p” pode receber o valor armazenado em “x”;
- Nesse caso, “x” aponta para o mesmo vetor que a variável “x”;
- O vetor pode ser acessado através do ponteiro também;
- $x[i]$  é o mesmo que  $*(p+i)$ .



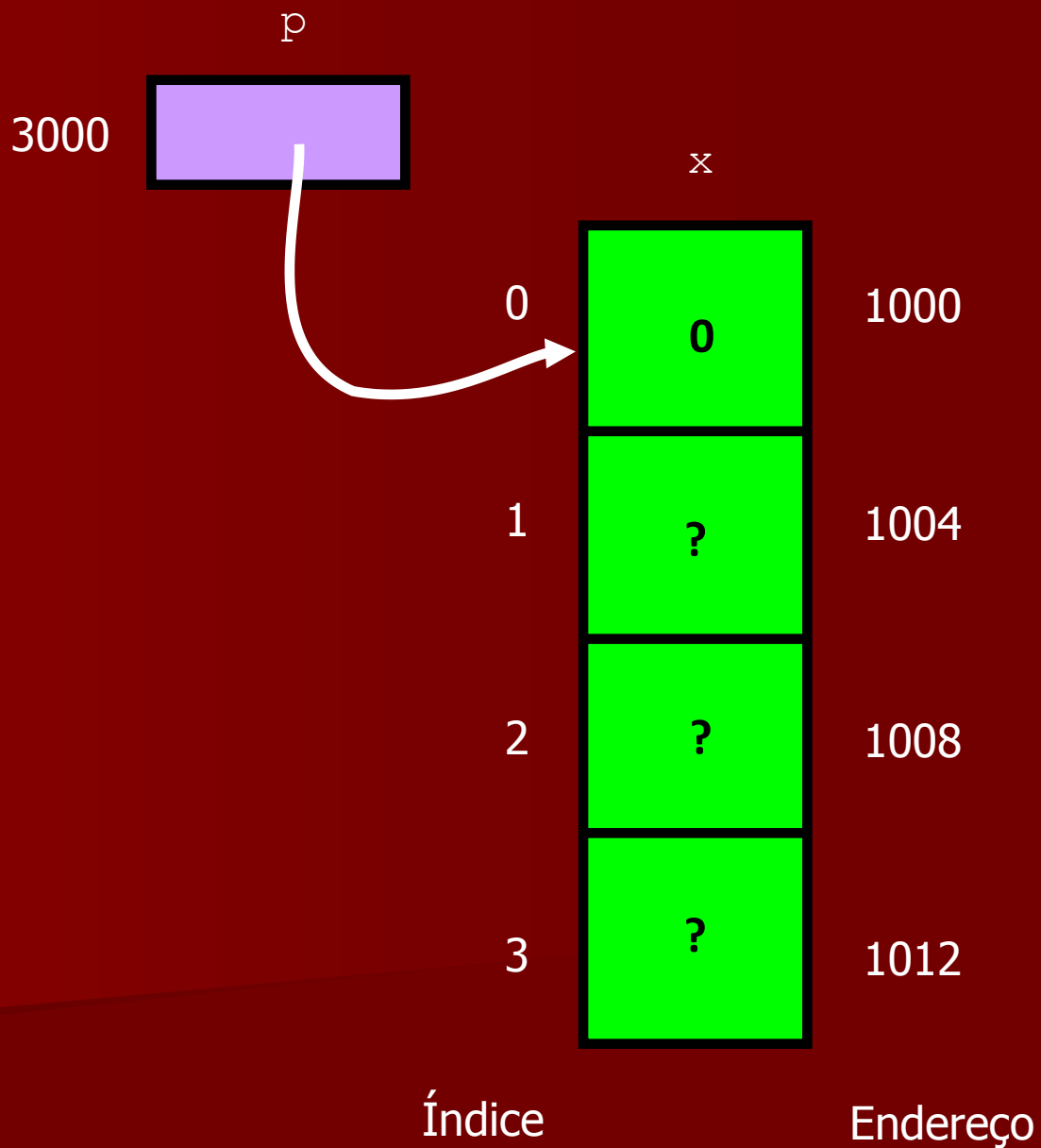
Índice

Endereço

`p=x;`

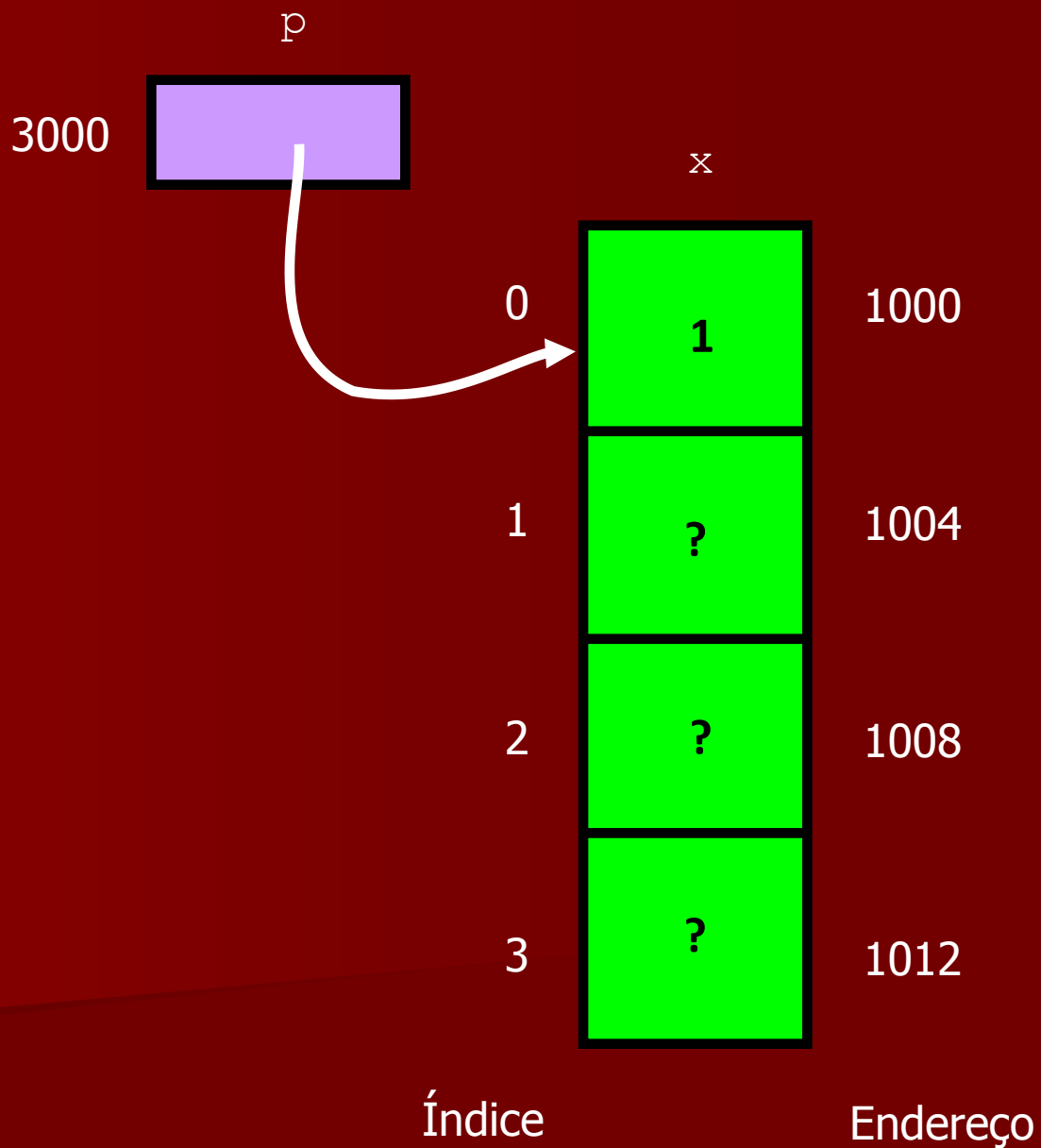


```
p=x;  
*p=0;
```

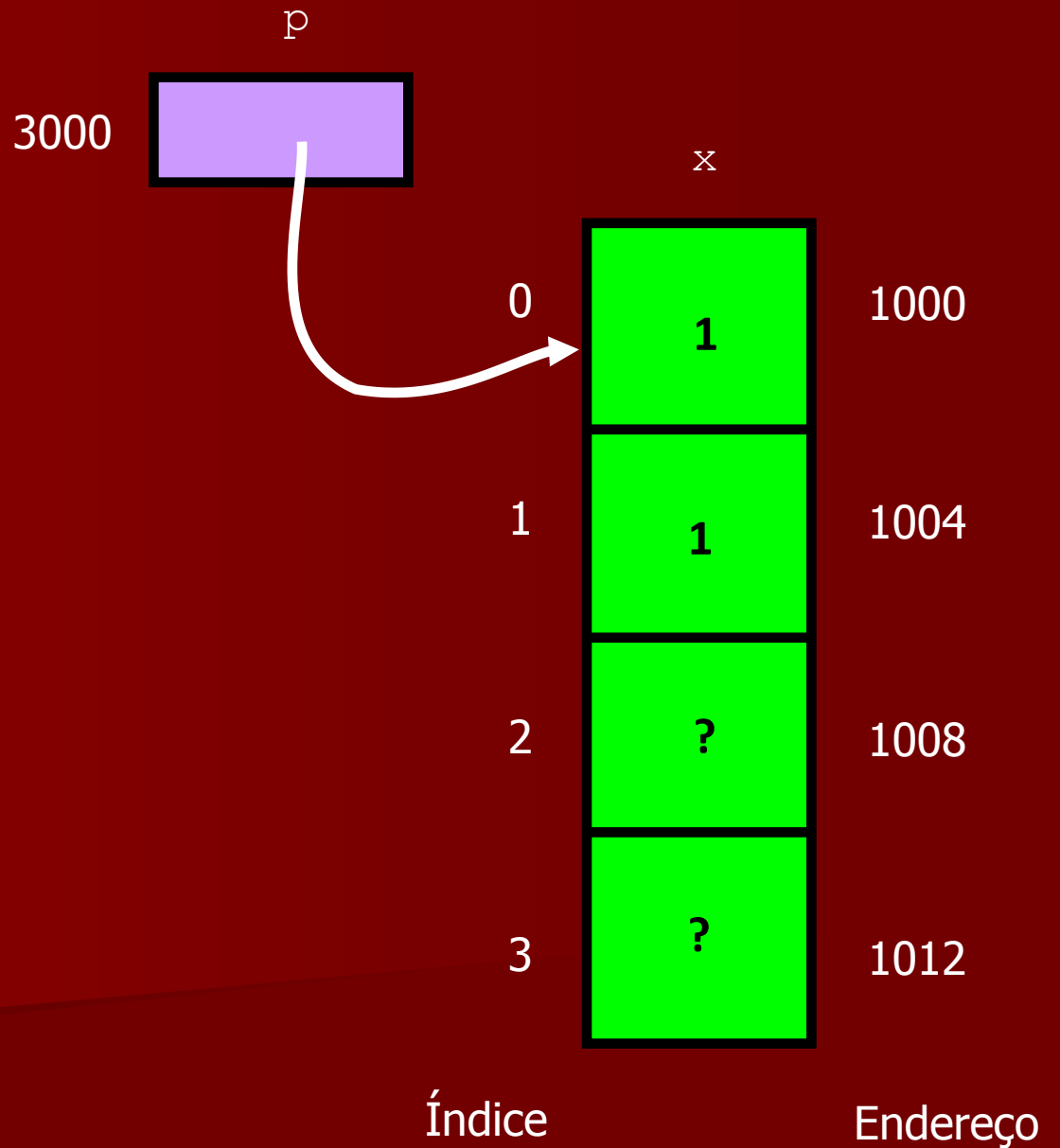




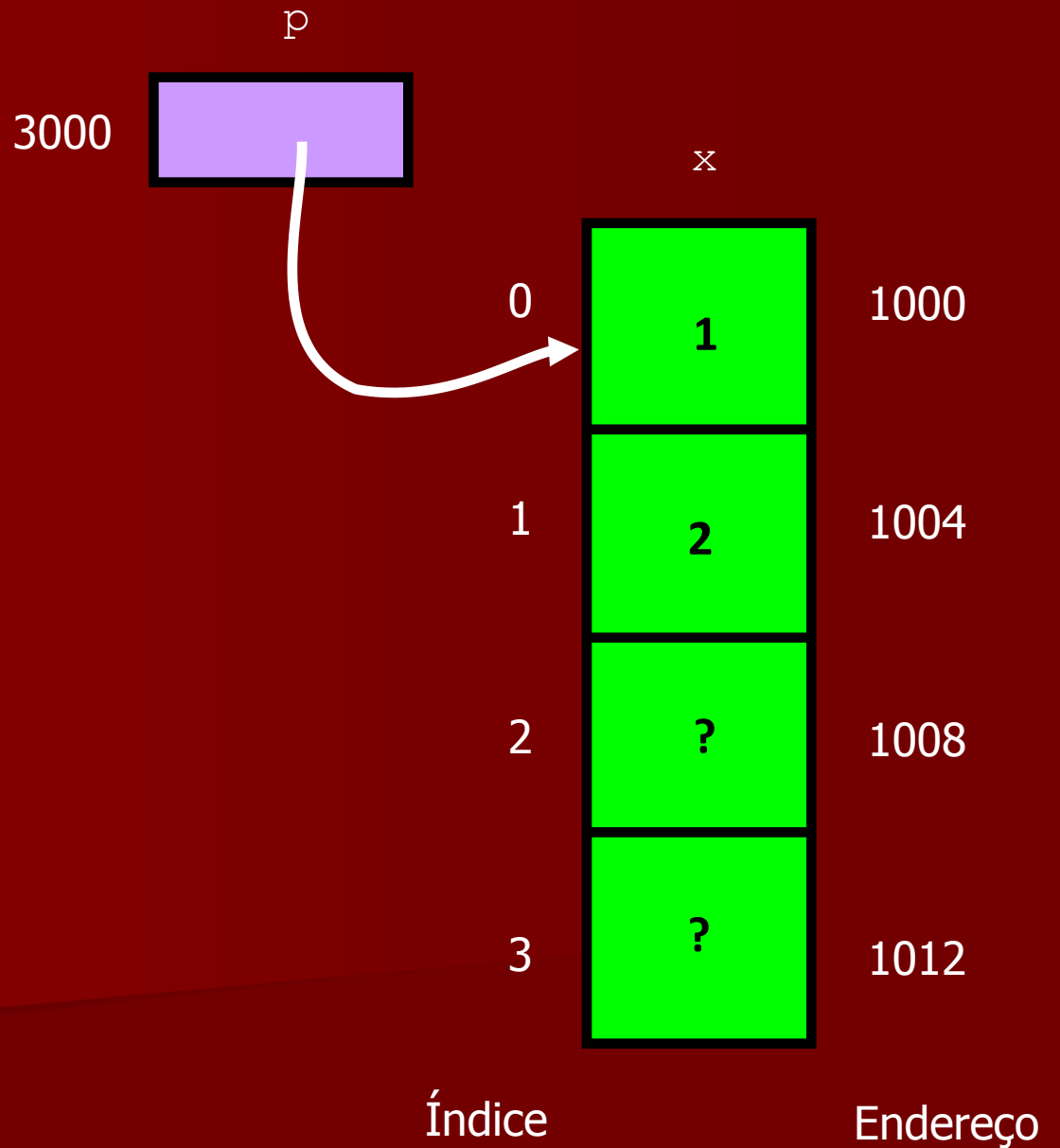
```
p=x;  
*p=0;  
*p=*p+1;
```



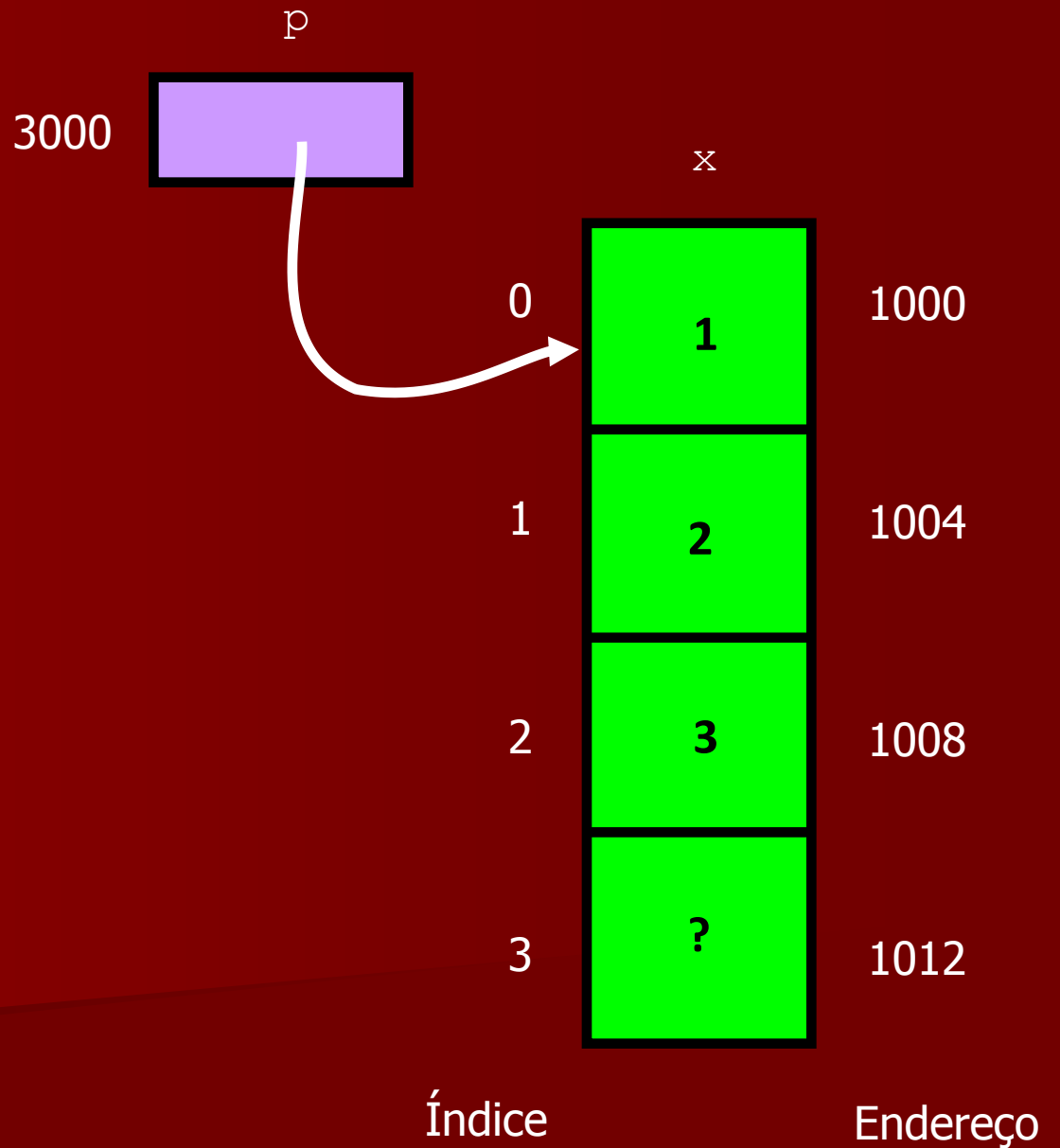
```
p=x;  
*p=0;  
*p=*p+1;  
*(p+1)=1;
```



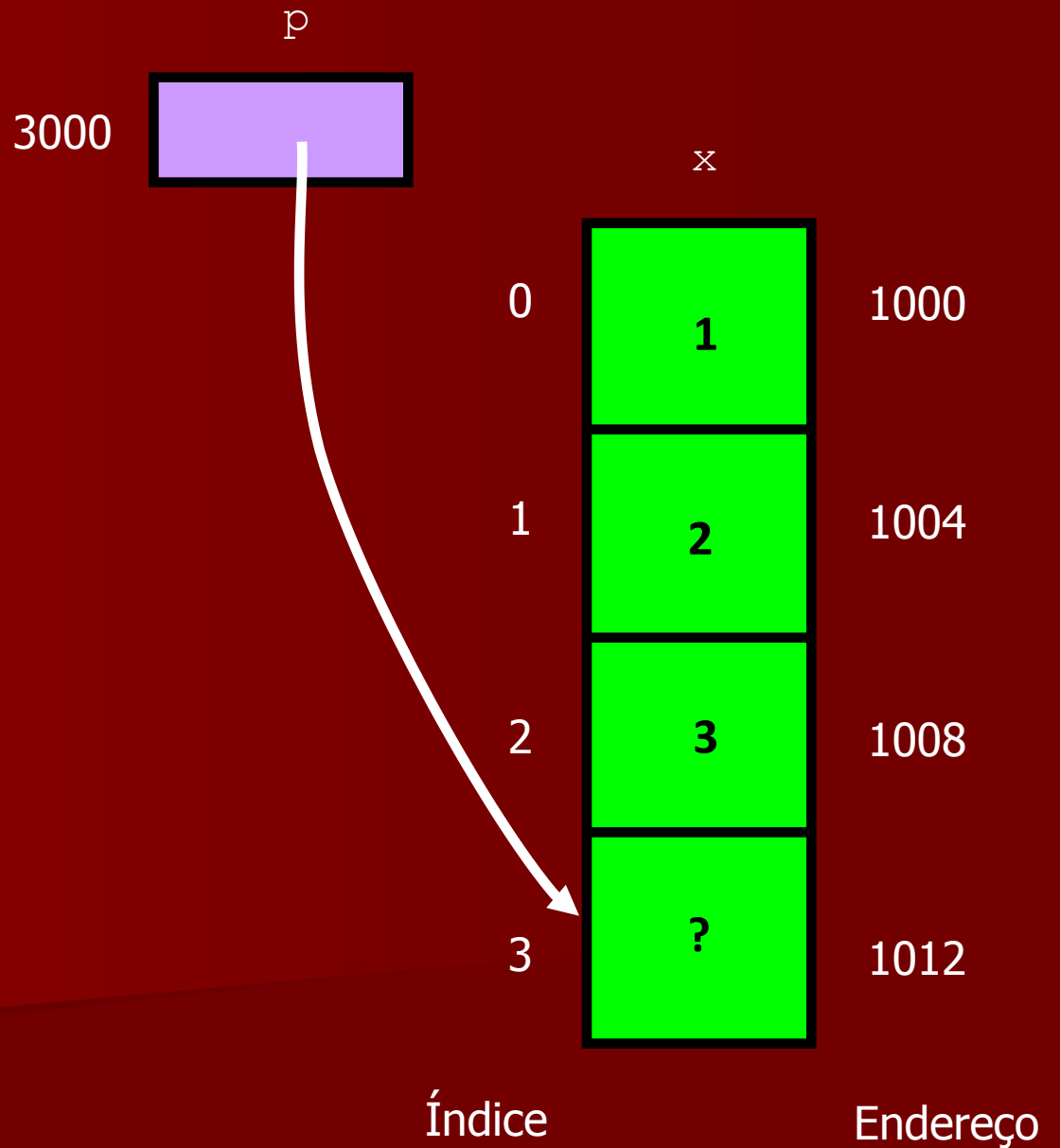
```
p=x;  
*p=0;  
*p=*p+1;  
*(p+1)=1;  
*(p+1)=*(p+1)+1;
```



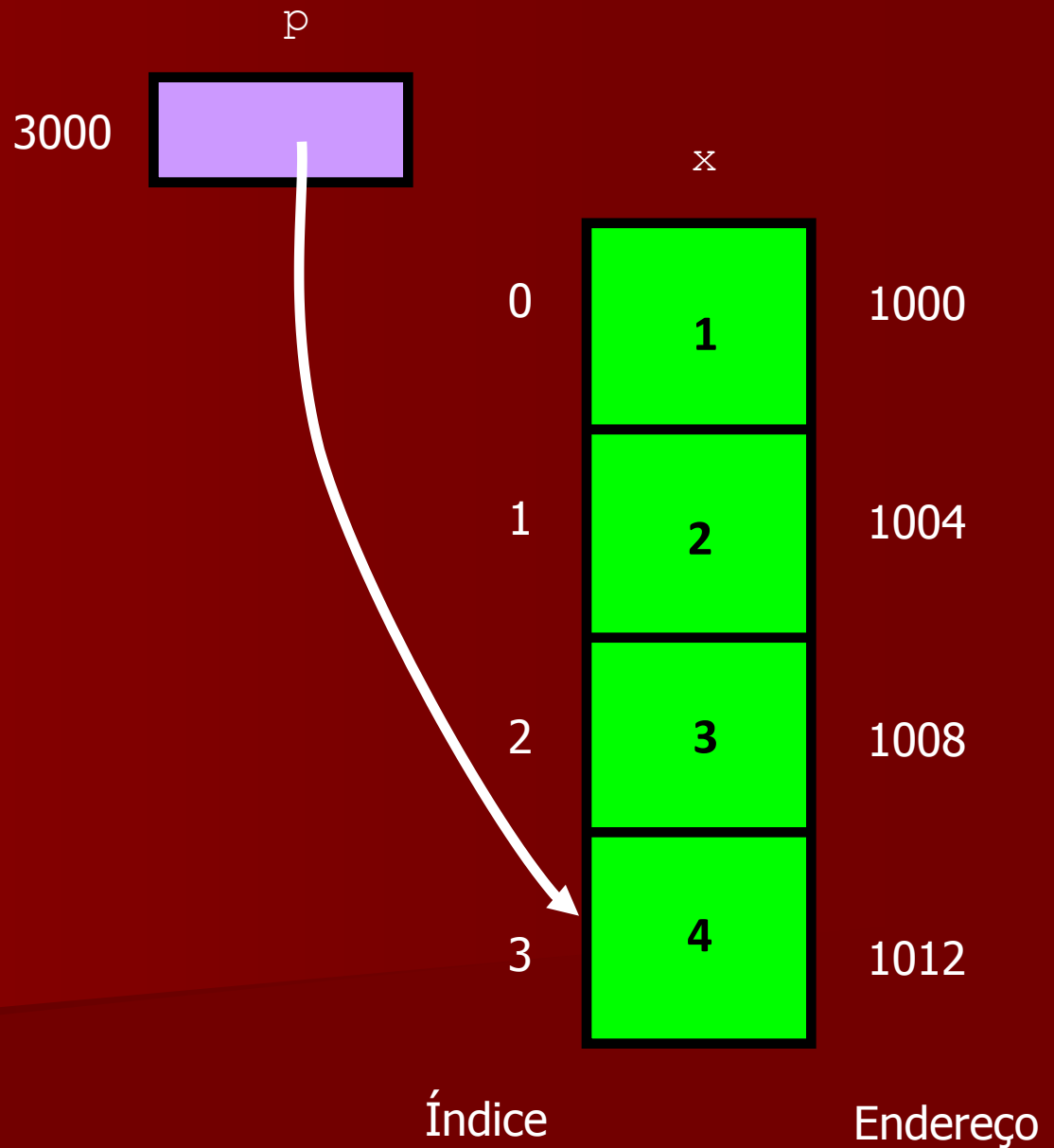
```
p=x;  
*p=0;  
*p=*p+1;  
*(p+1)=1;  
*(p+1)=*(p+1)+1;  
*(p+2)=*(p+1)+1;
```



```
p=x;  
*p=0;  
*p=*p+1;  
*(p+1)=1;  
*(p+1)=*(p+1)+1;  
*(p+2)=*(p+1)+1;  
p=p+3;
```

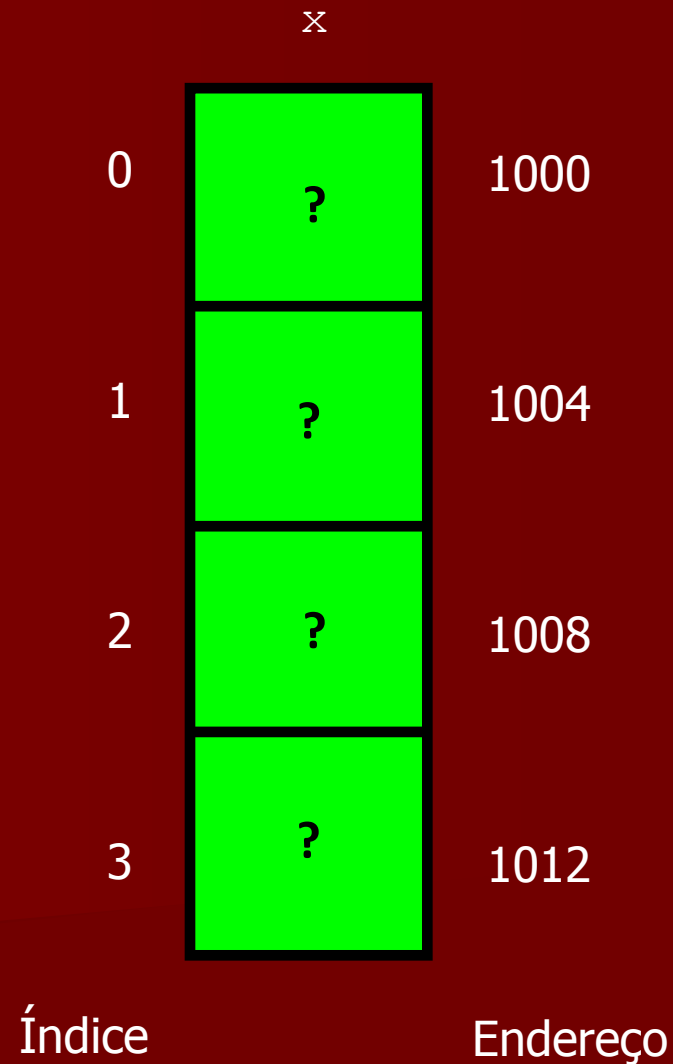


```
p=x;  
*p=0;  
*p=*p+1;  
*(p+1)=1;  
*(p+1)=*(p+1)+1;  
*(p+2)=*(p+1)+1;  
p=p+3;  
*p=4;
```

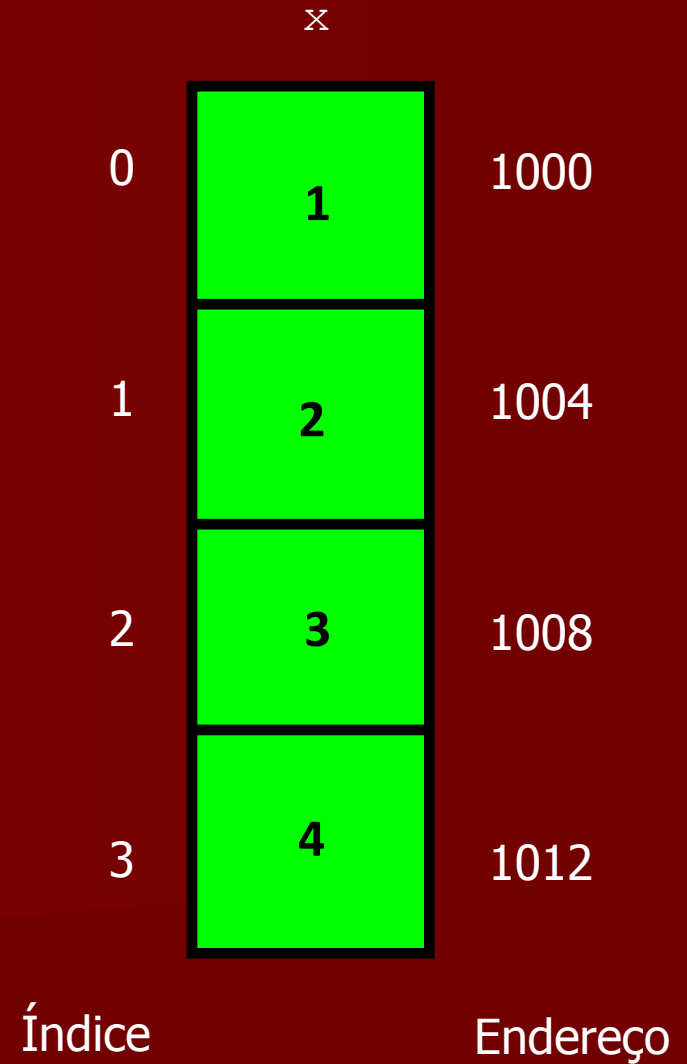


```
int x[4];
```

- “x”, que é o nome de um vetor de inteiros, pode ser usado da mesma forma que um ponteiro para inteiro;
- $x[i]$  é o mesmo que  $*(x+i)$ .



```
*x=0 ;  
*x=*x+1 ;  
*(x+1)=1 ;  
*(x+1)=*(x+1)+1 ;  
*(x+2)=*(x+1)+1 ;  
*(x+3)=4 ;
```





```
int i,x[4];  
for (i=0;i<4;i++)  
    *(x+i)=i+2;  
for (i=0;i<4;i++)  
    printf ("%d",*(x+i));
```

