Aula 3

Implementação de grafos por matriz de adjacência

Profa. Ariane Machado Lima

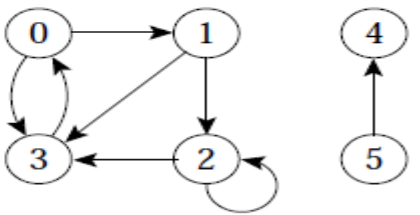
Aula passada...

- Noções básicas de grafos e exemplos de aplicações
- Direcionados x não-direcionados
- Vértices: grau, adjacentes, vizinhos
- Caminhos, ciclos

Ciclos

- Em um grafo direcionado:
  - Um caminho $(v_0, v_1, \ldots, v_k)$ forma um ciclo se $v_0 = v_k$ e o caminho contém pelo menos uma aresta.
  - O ciclo é simples se os vértices $v_1, v_2, \ldots, v_k$ são distintos.
  - O *self-loop* é um ciclo de tamanho 1.
  - Dois caminhos $(v_0, v_1, \ldots, v_k)$ e $(v'_0, v'_1, \ldots, v'_k)$ formam o mesmo ciclo se existir um inteiro j tal que $v'_i = v_{(i+j) \bmod k}$ para $i = 0, 1, \ldots, k-1$.

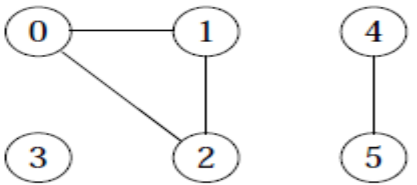
Ex.: O caminho $(0, 1, 2, 3, 0)$ forma um ciclo. O caminho$(0, 1, 3, 0)$ forma o mesmo ciclo que os caminhos $(1, 3, 0, 1)$ e $(3, 0, 1, 3)$.



Ciclos

- Em um grafo não direcionado:
  - Um caminho $(v_0, v_1, \ldots, v_k)$ forma um ciclo se $v_0 = v_k$ e o caminho contém pelo menos três arestas.
  - O ciclo é simples se os vértices $v_1, v_2, \ldots, v_k$ são distintos.

Ex.: O caminho $(0, 1, 2, 0)$ é um ciclo.



Aula de hoje

- Mais um último conceito que faltou (grafo ponderado)
- Implementação de grafo

Ex: Logística

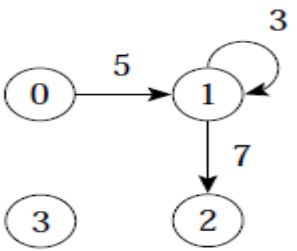
Como fazer rotas de distribuição?

https://blog.longa.com.br/roteirizacao-logistica/

- O que é um caminho?
- Quero repetir lugares?
- Quero voltar ao ponto de partida ao final?
- Qualquer caminho serve?

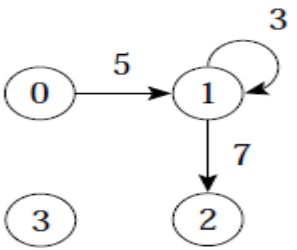
Grafo ponderado

possui pesos associados às arestas.



Grafo ponderado

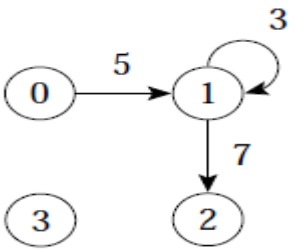
possui pesos associados às arestas.



Confusão entre os termos “caminho mais curto” e “caminho de peso/custo mínimo”

Grafo ponderado

possui pesos associados às arestas.



Confusão entre os termos “caminho mais curto” e “caminho de peso/custo mínimo”

Comprimento de um caminho é o número de arestas do caminho.
Peso ou custo de um caminho é a soma dos pesos/custos das arestas desse caminho

Implementações

O Tipo Abstratos de Dados Grafo

- Importante considerar os algoritmos em grafos como **tipos abstratos de dados**.
- Conjunto de operações associado a uma estrutura de dados.
- Independência de implementação para as operações.

Operações

- Que operações precisaríamos?

- Assuma o tipo **bool** “primitivo” e os tipos abstratos já definidos para **Grafo** e **Vertice**

Exemplos....

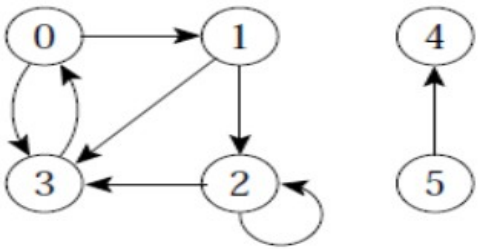
Operadores do TAD Grafo

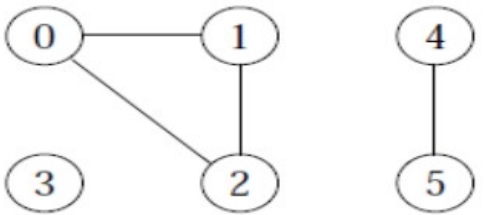
1. *InicializaGrafoVazio(Grafo)*: Inicializa um grafo vazio (sem arestas)
2. *InsereAresta(V1, V2, Peso, Grafo)*: Insere uma aresta no grafo.
3. *ExisteAresta(V1, V2, Grafo)*: Verifica se existe uma determinada aresta.
4. Obtem a lista de vértices adjacentes a um determinado vértice (tratada a seguir).
5. *RetiraAresta(V1, V2, Peso, Grafo)*: Retira uma aresta do grafo.
6. *LiberaGrafo(Grafo)*: Liberar o espaço ocupado por um grafo.
7. *ImprimeGrafo(Grafo)*: Imprime um grafo.
8. *GrafoTransposto(Grafo, GrafoT)*: Obtém o transposto de um grafo direcionado.
9. *RetiraMin(A)*: Obtém a aresta de menor peso de um grafo com peso nas arestas.

Operações sobre a lista de adjacentes

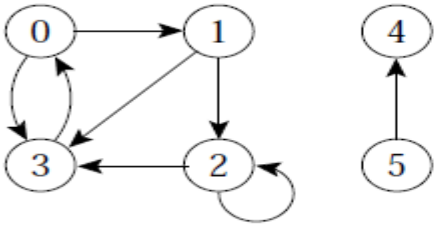
- Se está vazia
- Primeiro da lista
- Próximo da lista (iterador)

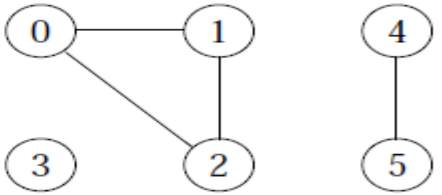
Como poderíamos implementar um grafo?





Matriz de Adjacência: Exemplo





	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

(a)

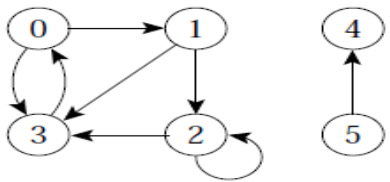
	0	1	2	3	4	5
0		1	1			
1	1		1			
2	1	1				
3						
4						1
5					1	

(b)

Matriz simétrica!
Poderíamos armazenar apenas
a parte triangular inferior ou superior,
mas não vamos fazer isso

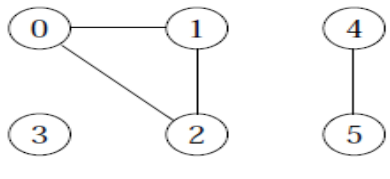
Matriz de Adjacência

- A matriz de adjacência de um grafo $G = (V, A)$ contendo n vértices é uma matriz $n \times n$ de *bits*, onde $A[i, j]$ é 1 (ou verdadeiro) se e somente se existe um arco do vértice i para o vértice j.
- Para grafos ponderados $A[i, j]$ contém o rótulo ou peso associado com a aresta e, neste caso, a matriz não é de *bits*.
- Se não existir uma aresta de i para j então é necessário utilizar um valor que não possa ser usado como rótulo ou peso.



	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

(a)



	0	1	2	3	4	5
0		1	1			
1	1		1			
2	1	1				
3						
4						1
5					1	

(b)

Pausa para dicas de programação (modularização em C)

- A sua estrutura de dados de grafos poderá ser necessária em vários programas diferentes

→ deveria estar encapsulada em um módulo

Para isso, criaremos um arquivo .h e um .c para implementar apenas a estrutura de dados de grafos por matriz de adjacência (e suas operações) (como fazemos com classes em Java).

Um outro programa .c irá testar esse módulo (implementação de grafo).

Matriz de Adjacência: Estrutura de Dados

Arquivo grafo_matrizadj.h

```
#include <stdbool.h>    /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100

typedef struct  {
  bool mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

Matriz de Adjacência: Estrutura de Dados

Arquivo grafo_matrizadj.h

```
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100

typedef struct  {
  bool mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

Alocação estática de memória

Você pode querer reutilizar esse código para diferentes tamanhos de grafos… numVertices define o grafo

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

Matriz de Adjacência: Estrutura de Dados

Arquivo grafo_matrizadj.h

```
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100

typedef struct  {
  bool mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

Alocação estática de memória

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

Você pode querer reutilizar esse código para diferentes tamanhos de grafos… numVertices define o grafo

```
/*
  ? inicializaGrafo(?):
*/
```

Matriz de Adjacência: Estrutura de Dados

Arquivo grafo_matrizadj.h

```
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100

typedef struct  {
  bool mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

Alocação estática de memória

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

Você pode querer reutilizar esse código para diferentes tamanhos de grafos… numVertices define o grafo

```
/*
  ? inicializaGrafo(?): Inicializa um grafo com nv vértices
  Preenche as células com 0 (representando ausência de aresta)
  Vértices vão de 1 a nv (ou 0 a nv-1).
  Retorna true se inicializou com sucesso e false c.c.
*/
```

Matriz de Adjacência: Estrutura de Dados

Arquivo grafo_matrizadj.h

```
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100

typedef struct  {
  bool mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

Alocação estática de memória

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

Você pode querer reutilizar esse código para diferentes tamanhos de grafos… numVertices define o grafo

```
/*
  bool inicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vértices
  Preenche as células com 0 (representando ausência de aresta)
  Vértices vão de 1 a nv (ou 0 a nv-1).
  Retorna true se inicializou com sucesso e false c.c.
*/
bool inicializaGrafo(Grafo* grafo, int nv);
```

Matriz de Adjacência: Estrutura de Dados

Arquivo grafo_matrizadj.h

```
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100

typedef struct  {
  bool mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

E se eu quiser um grafo ponderado? int, float, double, struct… ?

```
/*
  bool inicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vértices
  Preenche as células com 0 (representando ausência de aresta)
  Vértices vão de 1 a nv.
  Retorna true se inicializou com sucesso e false c.c.
*/
bool inicializaGrafo(Grafo* grafo, int nv);
```

Matriz de Adjacência: Estrutura de Dados

Arquivo grafo_matrizadj.h

```
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100

typedef struct  {
  bool mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

E se eu quiser um grafo ponderado? int, float, double, struct… ?

```
/*
  bool inicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vértices
  Preenche as células com 0 (representando ausência de aresta)
  Vértices vão de 1 a nv.
  Retorna true se inicializou com sucesso e false c.c.
*/bool inicializaGrafo(Grafo* grafo, int nv);
```

Matriz de Adjacência: Estrutura de Dados

Arquivo grafo_matrizadj.h

```
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100


typedef int Peso;
typedef struct {
  Peso mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

E se eu quiser um grafo ponderado? int, float, double, struct… ?

```
/*
  bool inicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vértices
  Preenche as células com 0 (representando ausência de aresta)
  Vértices vão de 1 a nv.
  Retorna true se inicializou com sucesso e false c.c.
*/bool inicializaGrafo(Grafo* grafo, int nv);
```

Matriz de Adjacência: Estrutura de Dados

Arquivo grafo_matrizadj.h

```
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100
#define AN -1  /* aresta nula, ou seja, valor que representa ausencia de aresta */

typedef int Peso;
typedef struct {
  Peso mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

E se eu quiser um grafo ponderado? int, float, double, struct… ?

```
/*
  bool inicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vértices
  Preenche as células com AN (representando ausência de aresta)
  Vértices vão de 1 a nv.
  Retorna true se inicializou com sucesso e false c.c.
*/
bool inicializaGrafo(Grafo* grafo, int nv);
```

Matriz de Adjacência: Estrutura de Dados

Arquivo grafo_matrizadj.h

```
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100
#define AN -1  /* aresta nula, ou seja, valor que representa ausencia de aresta */

typedef int Peso;
typedef struct  {
  Peso mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

Alocação estática de memória

- Velocidade de alocação (em tempo de compilação e não de execução)
- Menos problemas para o programador (*segmentation faults*…)

Você pode querer reutilizar esse código para diferentes tamanhos de grafos… → número de vértice é um parâmetro

Matriz de Adjacência: Estrutura de Dados

Arquivo grafo_matrizadj.h

```
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100
#define AN -1  /* aresta nula, ou seja, valor que representa ausencia de aresta */

typedef int Peso;
typedef struct {
  Peso mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

Flexibilidade no tipo de aresta

```
/*
  bool inicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vértices
  Preenche as células com AN (representando ausência de aresta)
  Vértices vão de 1 a nv.
  Retorna true se inicializou com sucesso e false c.c.
*/bool inicializaGrafo(Grafo* grafo, int nv);
```

Arquivo grafo_matrizadj.c

```c
#include <stdio.h>
#include "grafo_matrizadj.h"

/*
  InicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vertices
  Vertices vao de 1 a nv.
  Preenche as celulas com AN (representando ausencia de aresta)
  Retorna true se inicializou com sucesso e false caso contrario
*/
bool inicializaGrafo(Grafo* grafo, int nv)
```

Como você implementaria?

Abre parêntesis...

Passagem por valor x passagem por referência

Variáveis em C

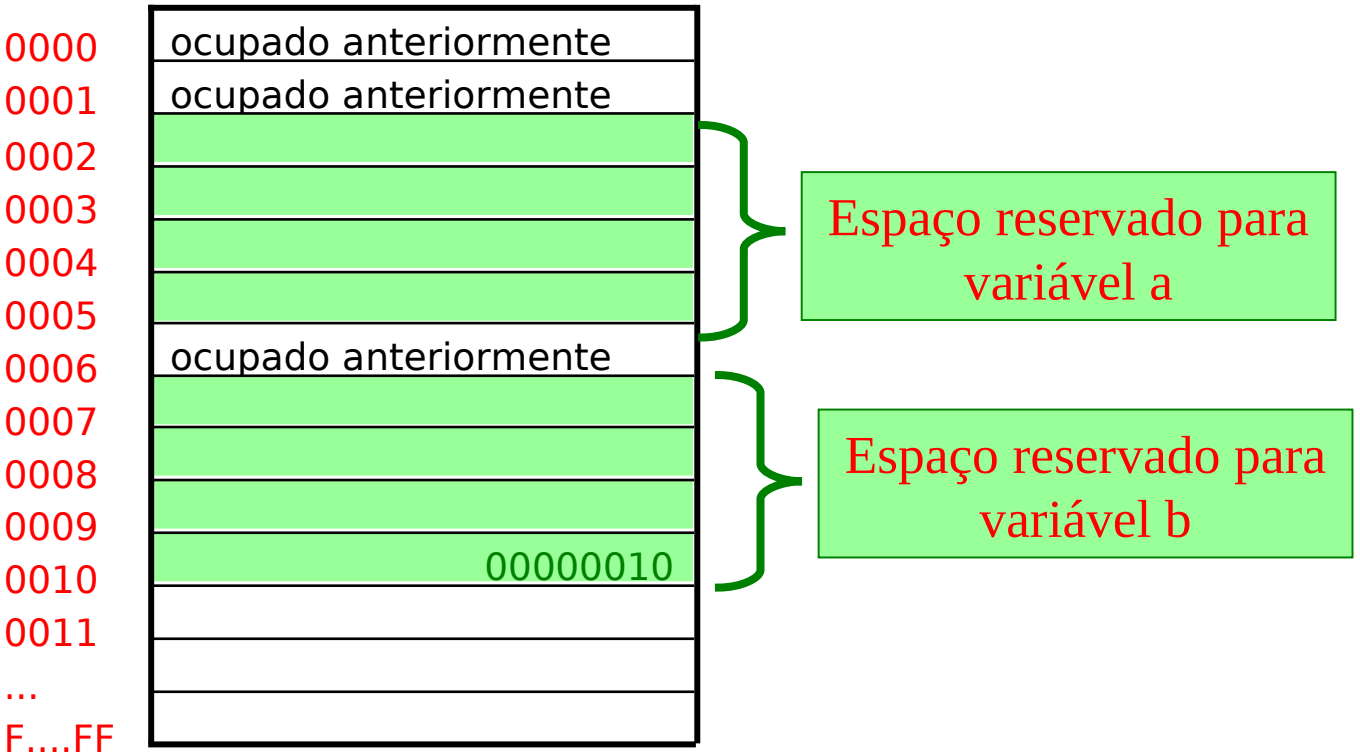
- Importante:
  - Dizemos que a e b armazenam diretamente o seu conteúdo

```
int a;
```

Variáveis em C

- Importante:
  - Dizemos que a e b armazenam diretamente o seu conteúdo

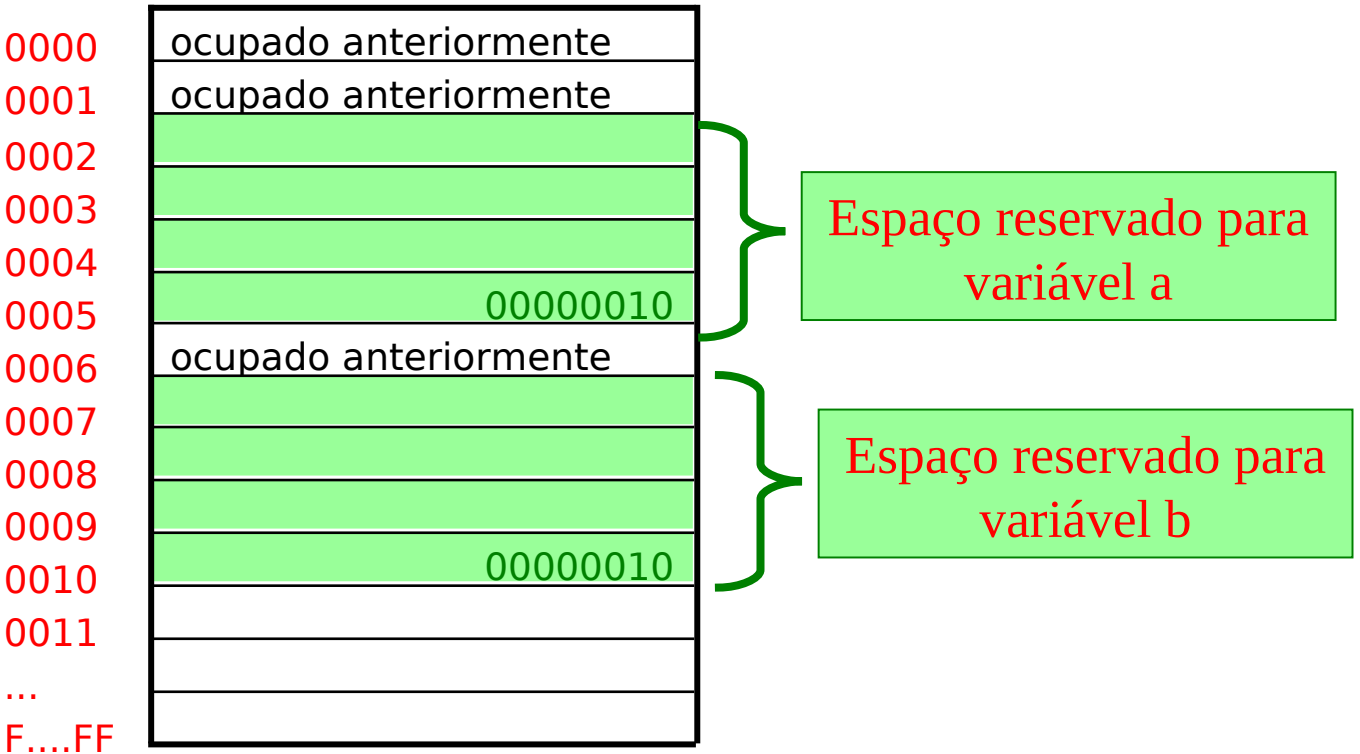
```
int a;

int b = 2;
```



Variáveis em C

- Importante:
  - Dizemos que a e b armazenam diretamente o seu conteúdo

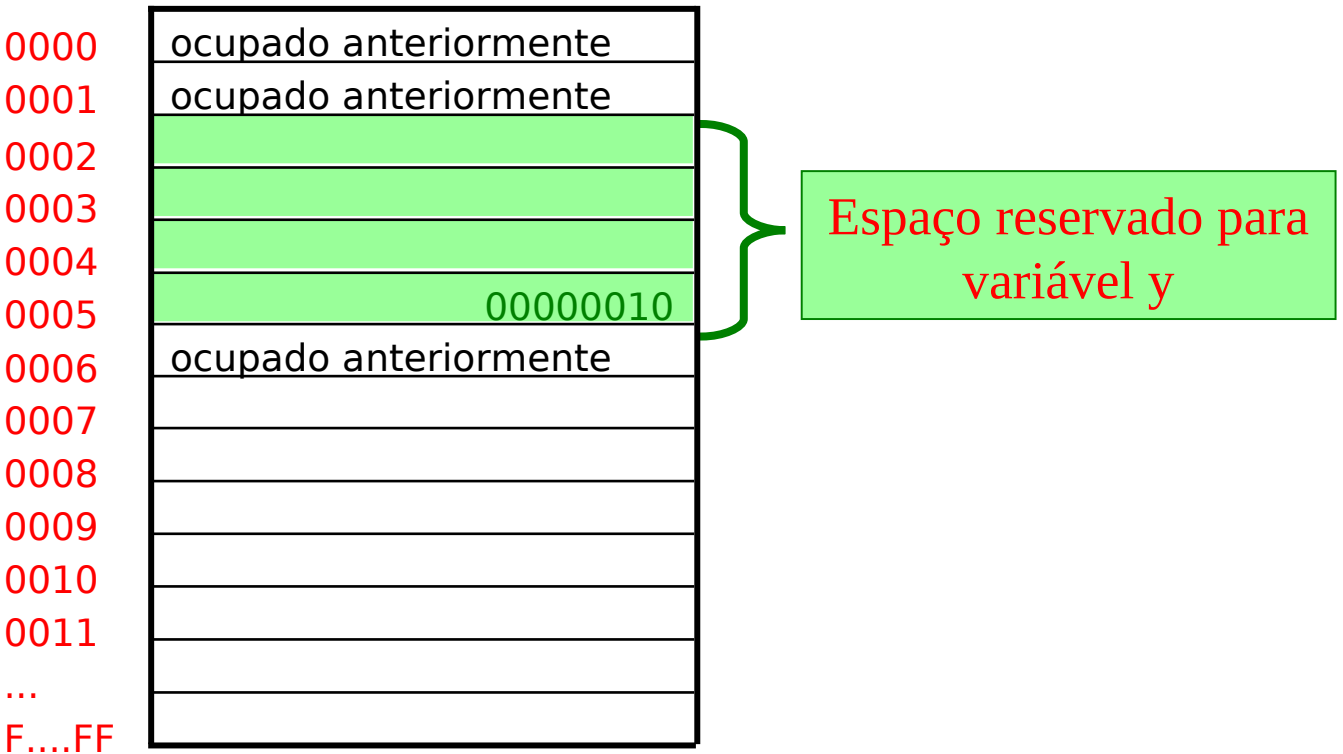
```
int a;

int b = 2;

a = b;
```



Variáveis em C

```
void fazAlgo (int x)
{
        (…)
        x++;
        (…)
}
```

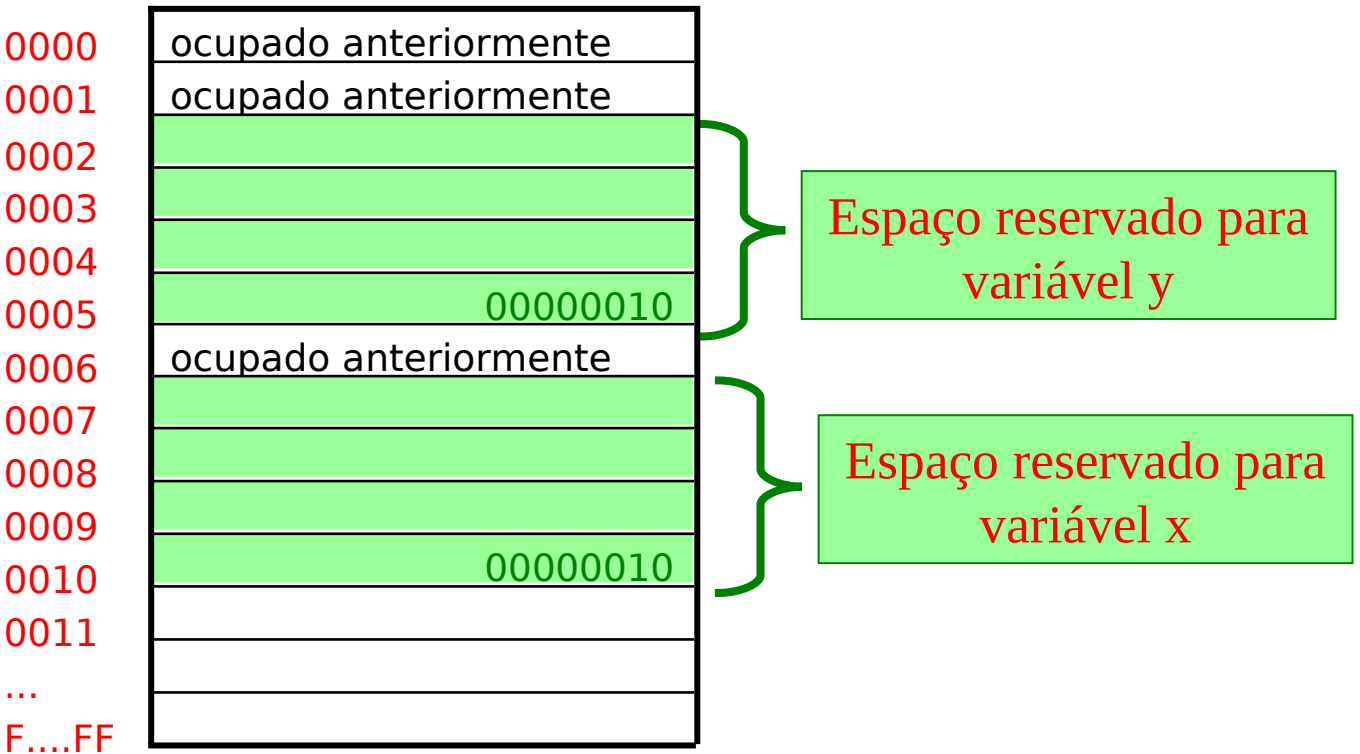
```
int y = 2;

fazAlgo(y);
```



Variáveis em C

```
void fazAlgo (int x)
{
        (…)
        x++;
        (…)
}
```

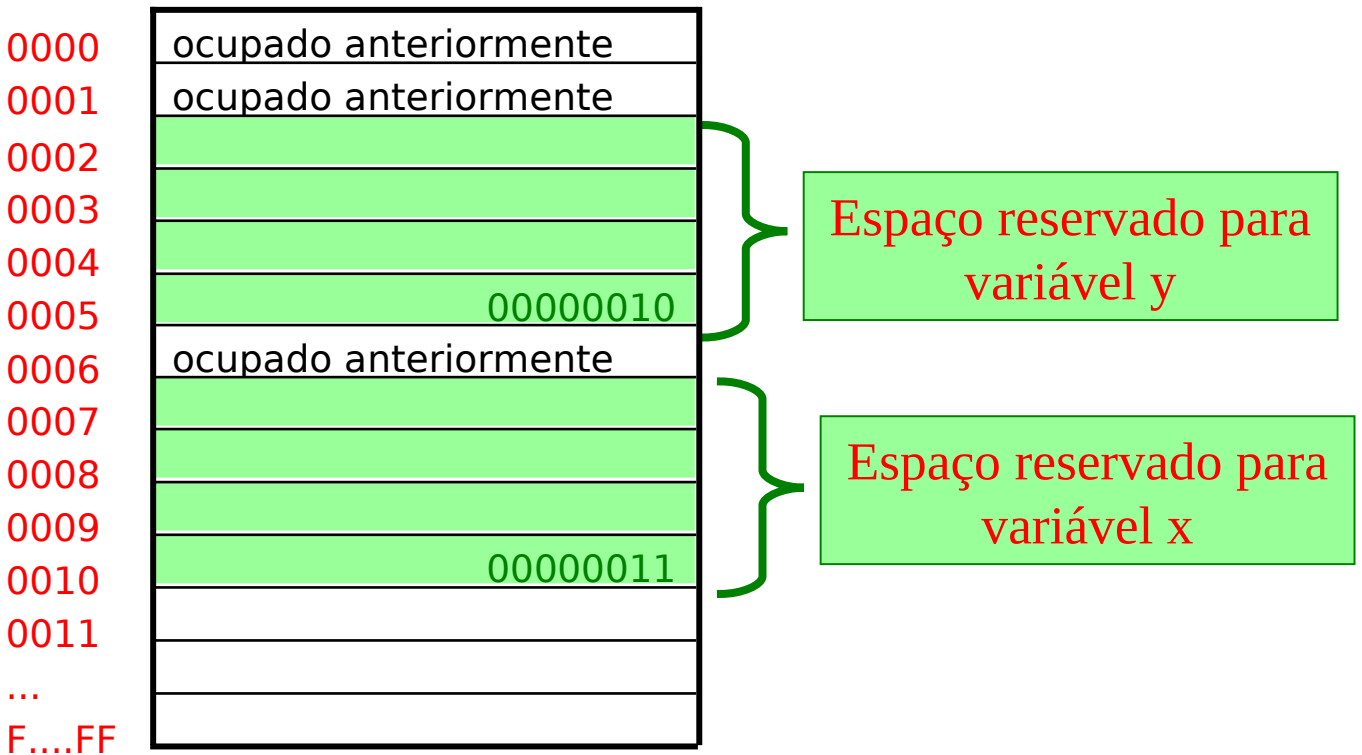
```
int y = 2;

fazAlgo(y);
```



Variáveis em C

```
void fazAlgo (int x)
{
        (…)
        x++;
        (…)
}
```

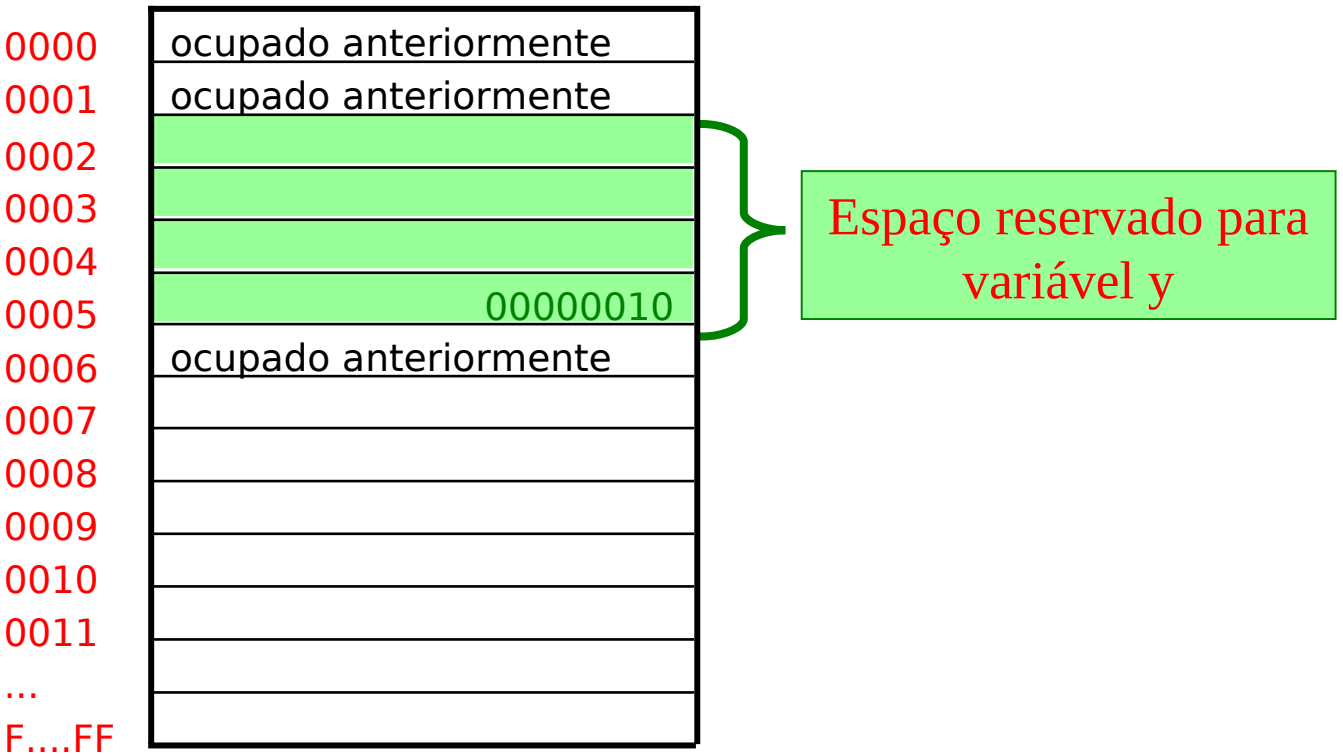
```
int y = 2;

fazAlgo(y);
```



Variáveis em C

```
void fazAlgo (int x)
{
        (…)
        x++;
        (…)
}
```

```
int y = 2;

fazAlgo(y);
```

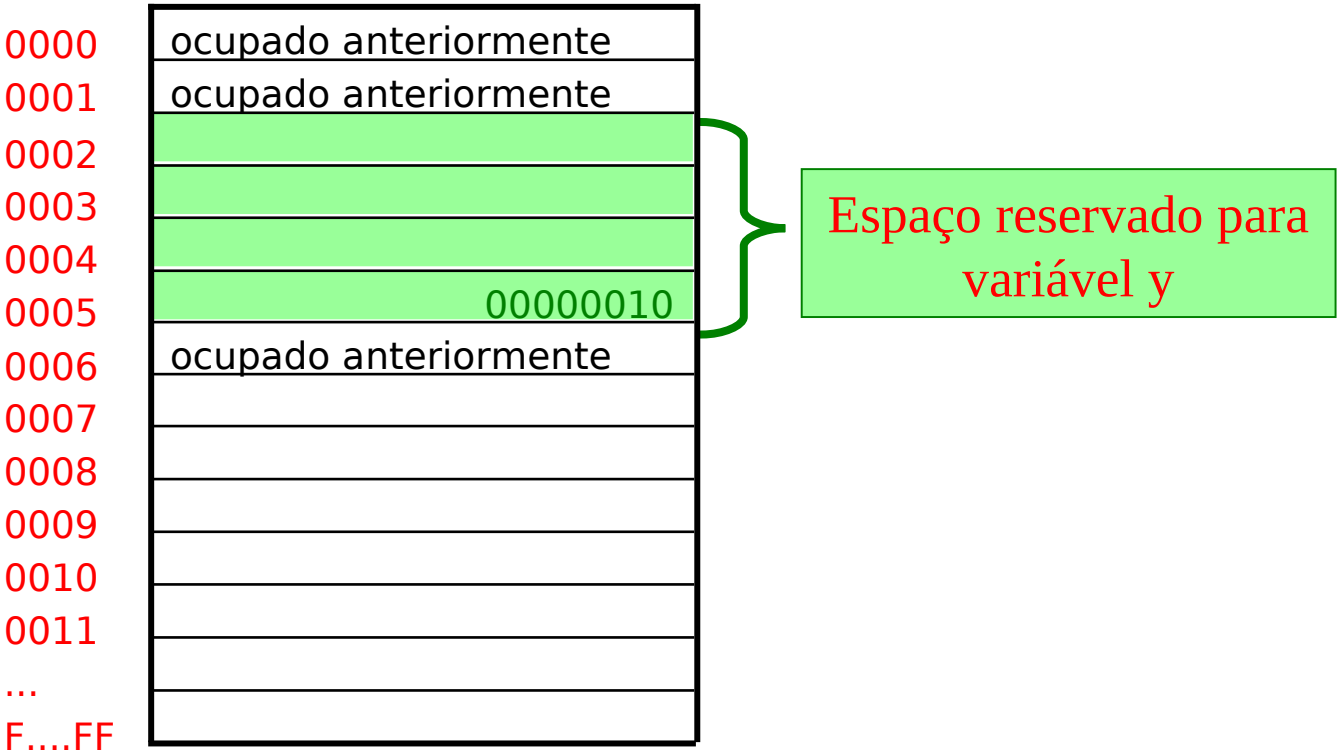


Variáveis em C

- Por isso: dizemos que a mudança em um parâmetro dentro de um método não se reflete fora dele: *variáveis locais*

```
void fazAlgo (int x)
{
        (…)
        x++;
        (…)
}
```

```
int y = 2;

fazAlgo(y);
```

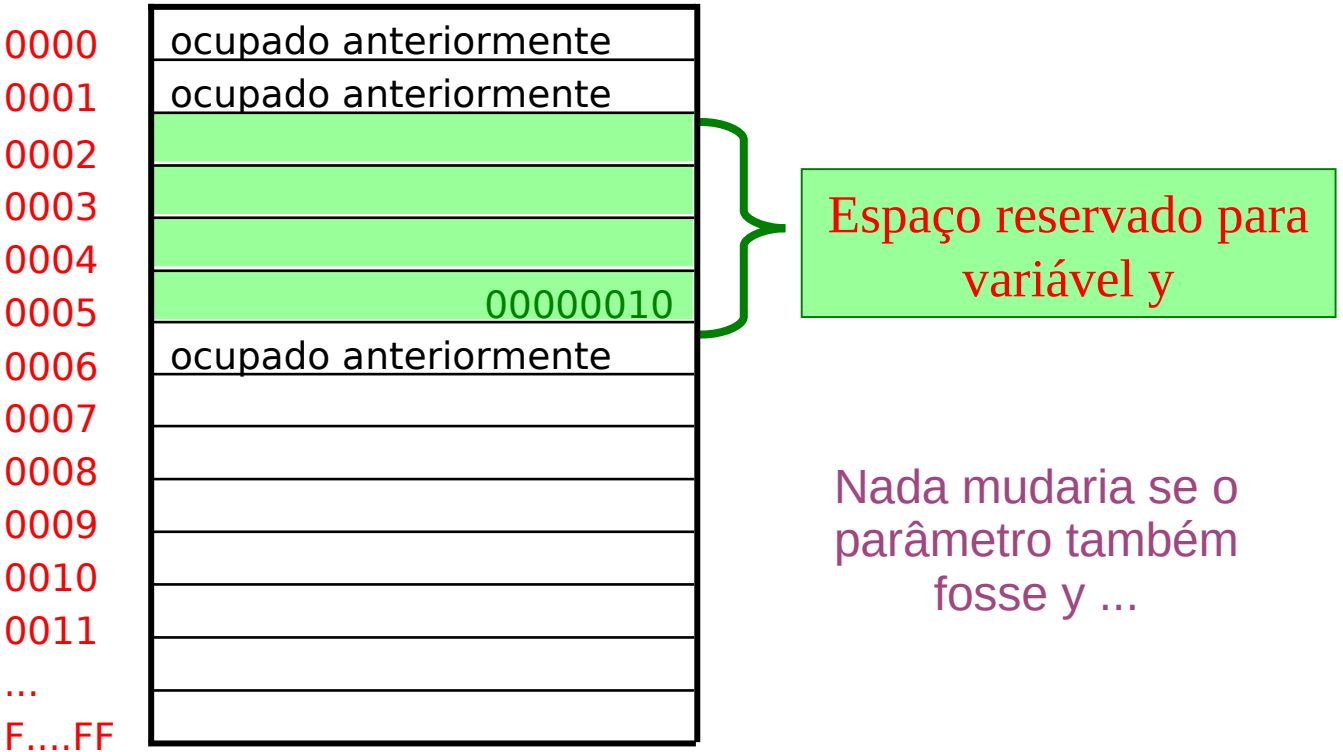


Variáveis em C

- Por isso: dizemos que a mudança em um parâmetro dentro de um método não se reflete fora dele: *variáveis locais*

```
void fazAlgo (int x)
{
        (…)
        x++;
        (…)
}
```

```
int y = 2;

fazAlgo(y);
```



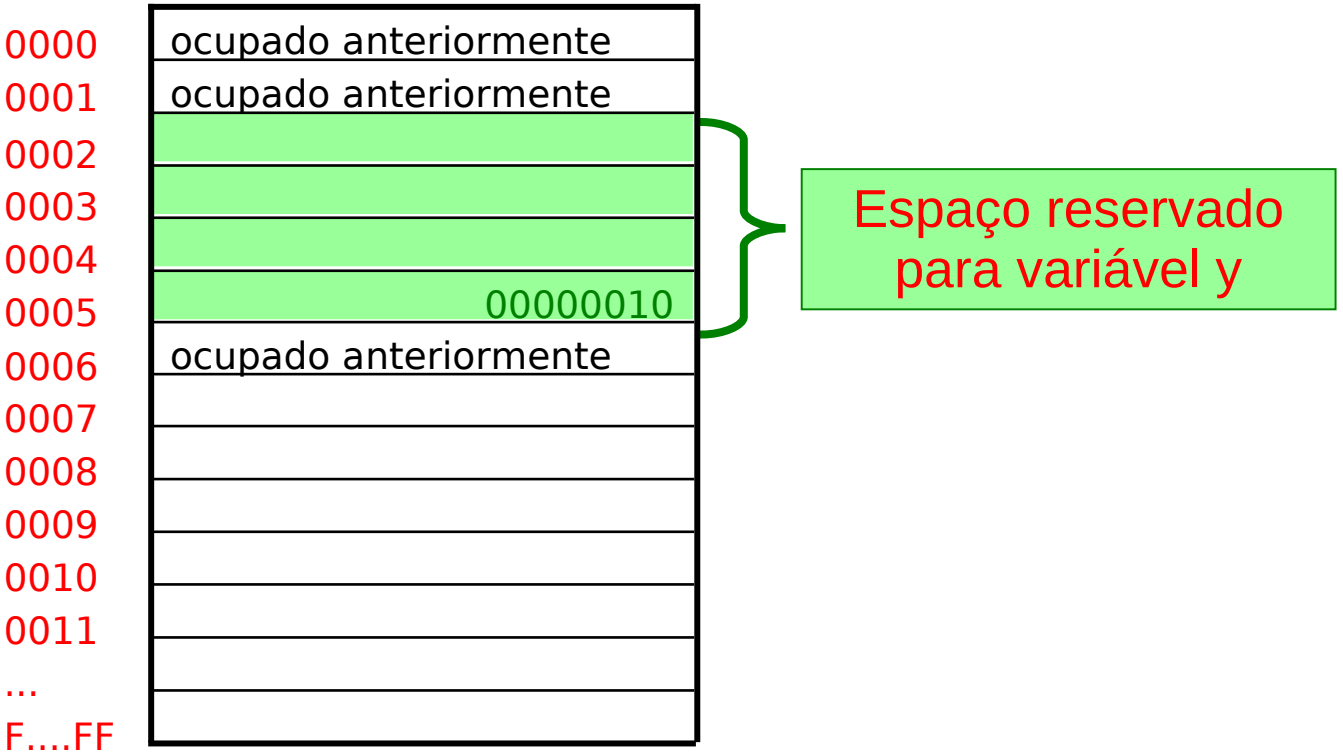
Nada mudaria se o parâmetro também fosse y ...

Variáveis em C

- Passagem de parâmetro por VALOR: o argumento é avaliado e COPIADO para o parâmetro

```
void fazAlgo (int x)
{
        (…)
        x++;
        (…)
}
```

```
int y = 2;

fazAlgo(y);
```



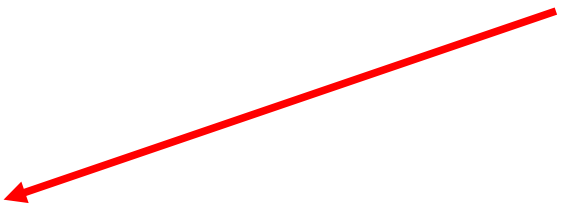
Variáveis em C

- Passagem de parâmetro por VALOR: o argumento é avaliado e COPIADO para o parâmetro

```
void fazAlgo (int x)
{
        (…)
        x++;
        (…)
}
```

```
int y = 2;

fazAlgo((y * 4) % 3);
```

Avaliado!!!!



NÃO em C, nem em Java!!!!! - Ex: C++ SIM

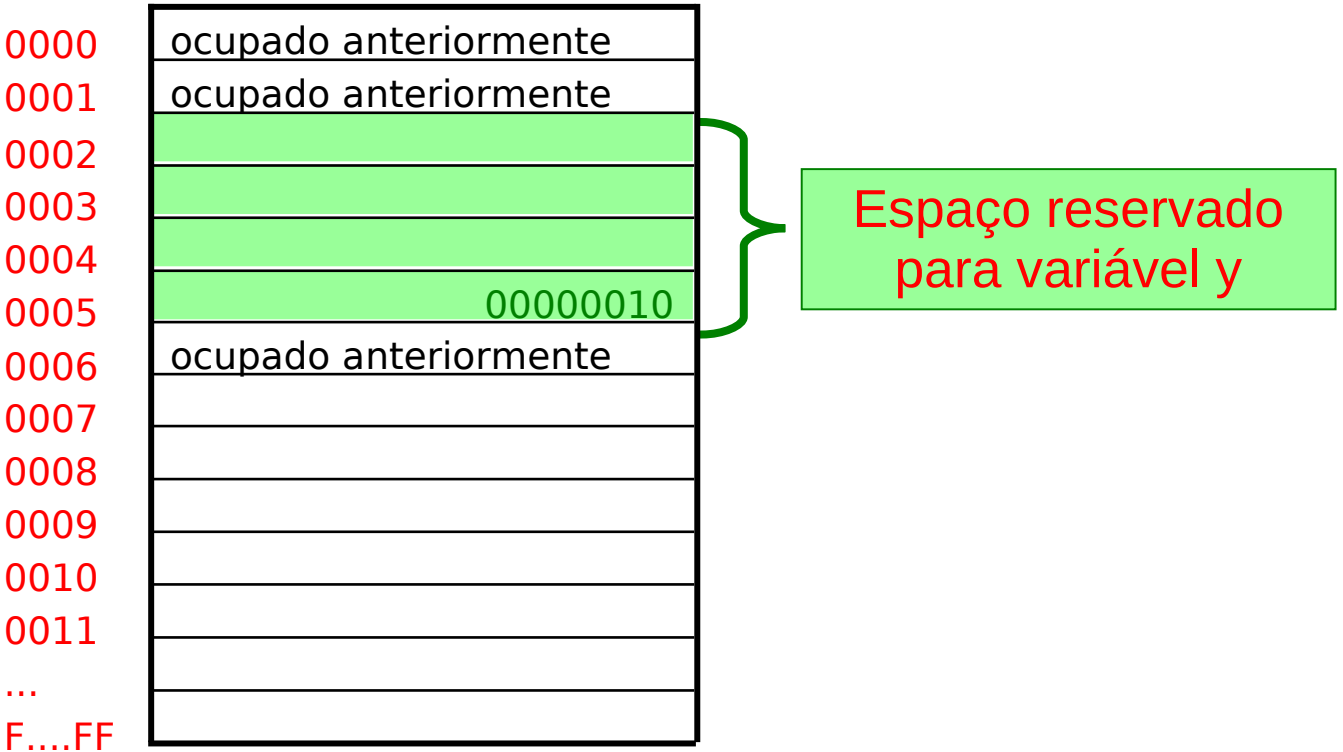
- Passagem de parâmetro por REFERÊNCIA: o parâmetro é um apelido (“atalho”) para o argumento (os dois apontam para o mesmo endereço em memória!!!)

NÃO em C, nem em Java!!!!! - Ex: C++ SIM

- Passagem de parâmetro por REFERÊNCIA: o parâmetro é um apelido (“atalho”) para o argumento (os dois apontam para o mesmo endereço em memória!!!)

```
void fazAlgo (int &x)
{
        (…)
        x++;
        (…)
}
```

```
int y = 2;

fazAlgo(y);
```

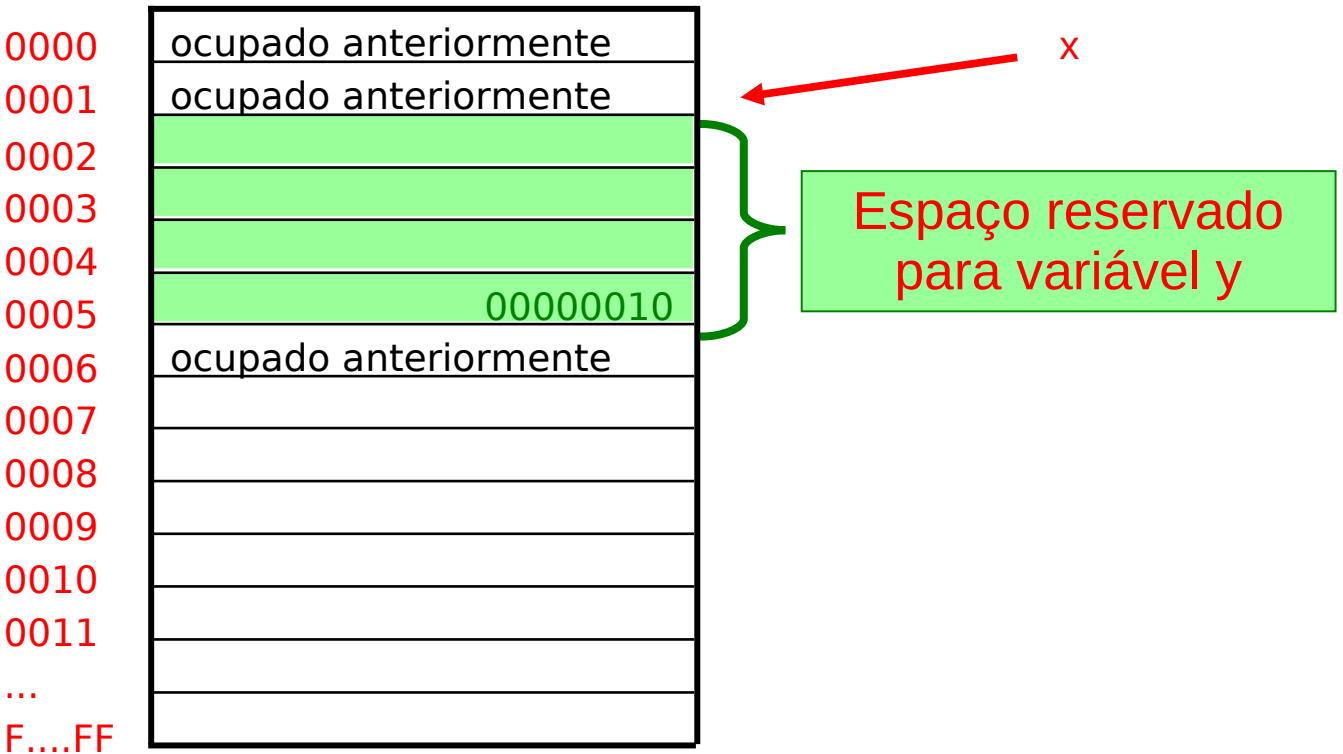


NÃO em C, nem em Java!!!!! - Ex: C++ SIM

- Passagem de parâmetro por REFERÊNCIA: o parâmetro é um apelido (“atalho”) para o argumento (os dois apontam para o mesmo endereço em memória!!!)

```
void fazAlgo (int &x)
{
        (…)
        x++;
        (…)
}
```

```
int y = 2;

fazAlgo(y);
```

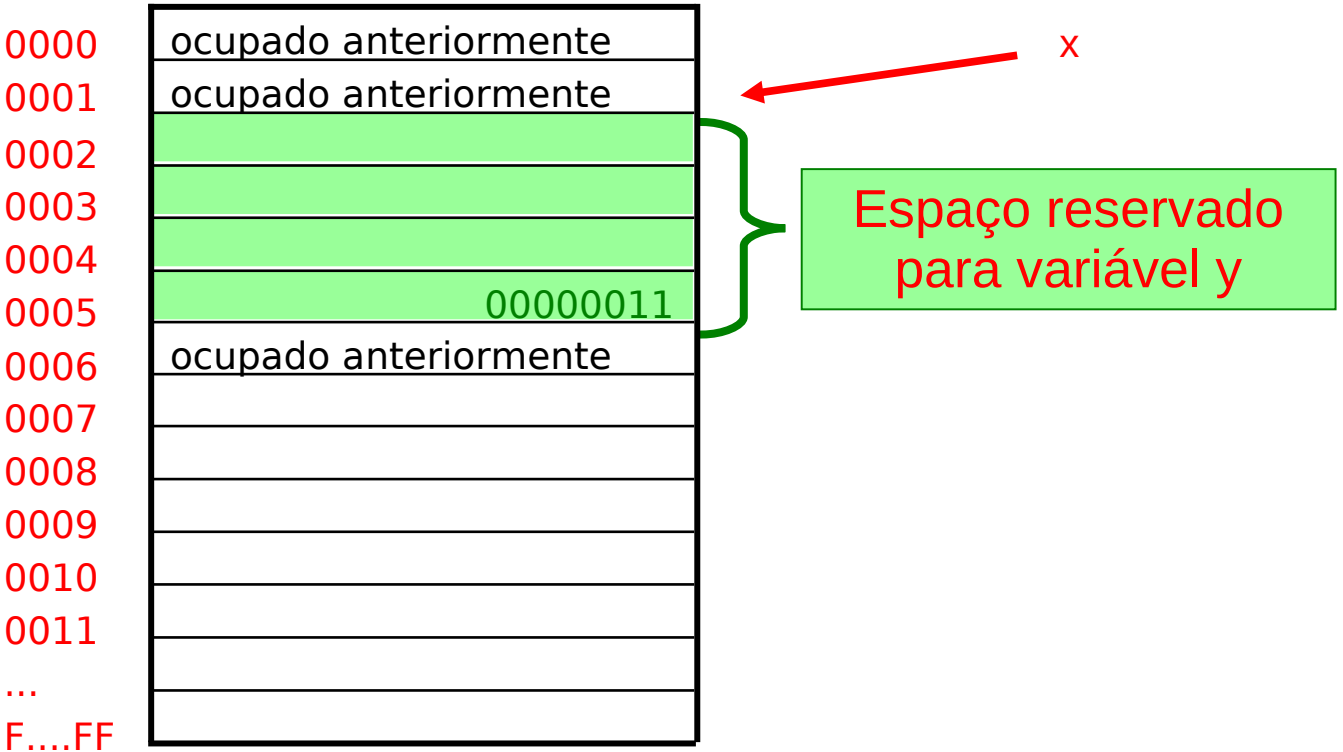


NÃO em C, nem em Java!!!!! - Ex: C++ SIM

- Passagem de parâmetro por REFERÊNCIA: o parâmetro é um apelido (“atalho”) para o argumento (os dois apontam para o mesmo endereço em memória!!!)

```
void fazAlgo (int &x)
{
        (…)
        x++;
        (…)
}
```

```
int y = 2;

fazAlgo(y);
```

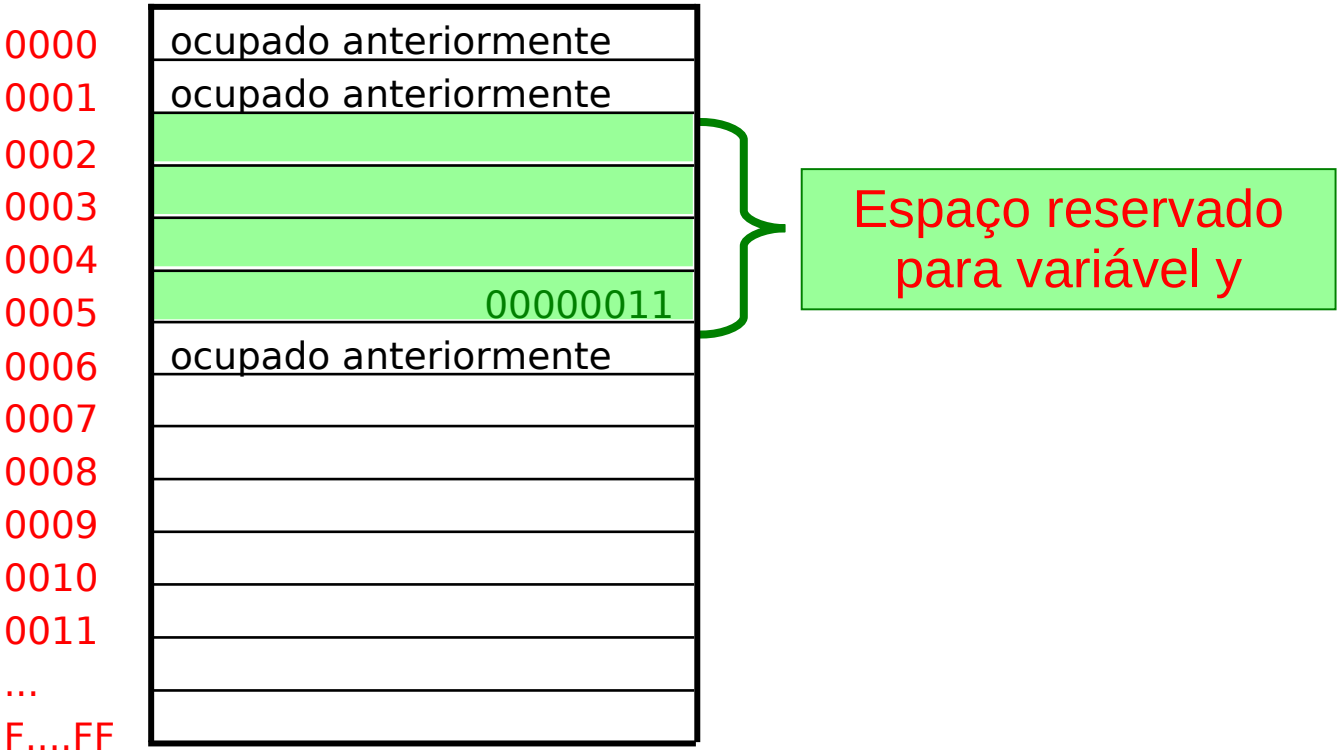


NÃO em C, nem em Java!!!!! - Ex: C++ SIM

- Passagem de parâmetro por REFERÊNCIA: o parâmetro é um apelido (“atalho”) para o argumento (os dois apontam para o mesmo endereço em memória!!!)

```
void fazAlgo (int &x)
{
        (…)
        x++;
        (…)
}
```

```
int y = 2;

fazAlgo(y);
```



NÃO em C, nem em Java!!!!! - Ex: C++ SIM

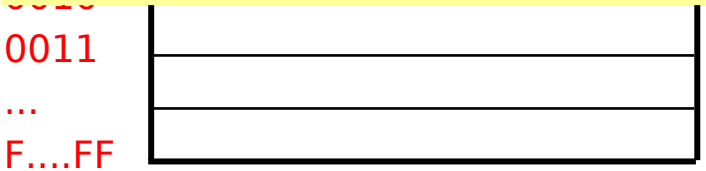
- Passagem de parâmetro por REFERÊNCIA: o parâmetro é um apelido (“atalho”) para o argumento (os dois apontam para o mesmo endereço em memória!!!)

```
void fazAlgo (int &x)
{
        (…)
        x++;
        (…)
}
```

```
int y = 2;

fazAlgo(y);
```

ISSO É C++, NÃO É C !!!

Em C, quando nós passamos ponteiro como parâmetro, nós obtemos o efeito de alterar o CONTEÚDO de uma variável dentro de uma função, mas se eu alterar o valor do parâmetro (ponteiro) eu acesso outro endereço de memória!!!



Em resumo

Passagem de parâmetros por valor: o parâmetro é uma cópia do argumento (o conteúdo é o mesmo, mas são dois espaços em memória distintos)

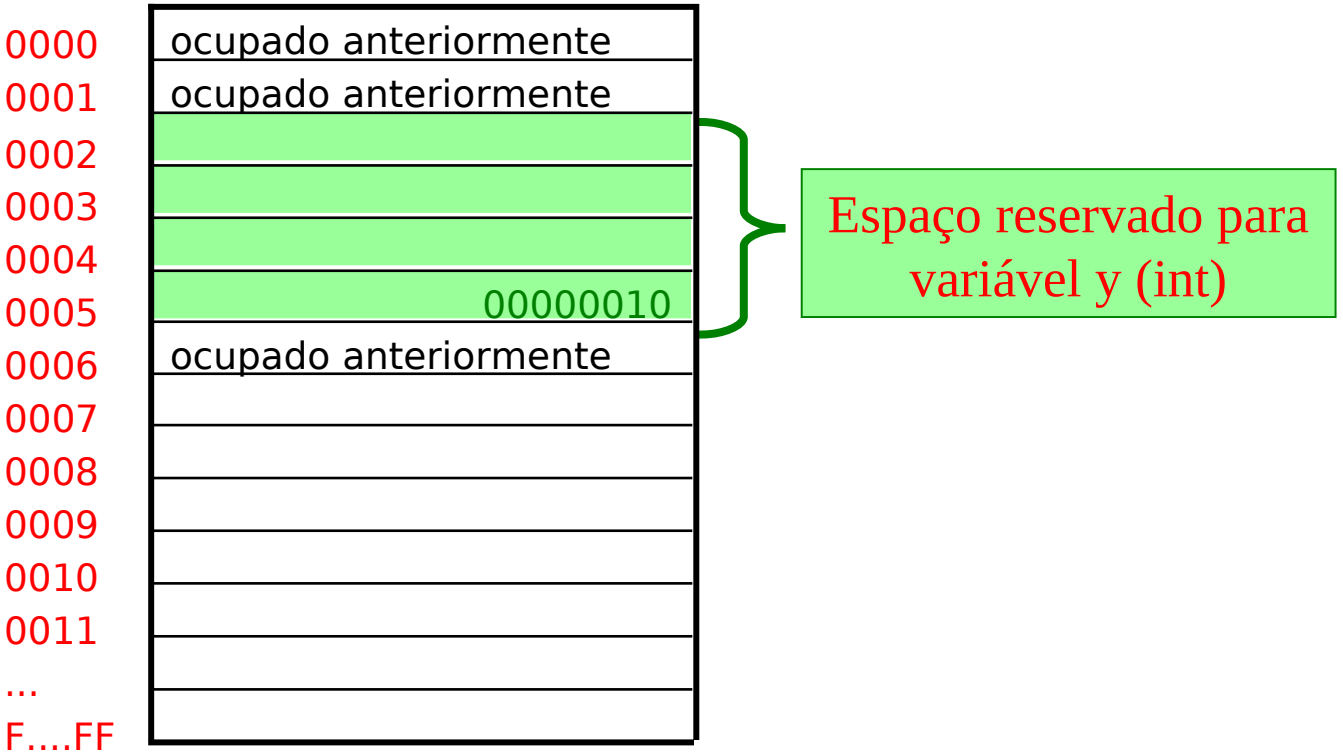
Passagem de parâmetros por referência: o parâmetro e o argumento correspondem ao mesmo espaço na memória

Como simular isso em C?

- Passando o endereço (ponteiro) da variável a ser alterada: o endereço é COPIADO (passagem por valor, afinal), mas aponta para o mesmo espaço em memória que contém a variável que você quer alterar!!!)

```
void fazAlgo (int* x)
{
        (…)
        (*x)++;
        (…)
}
```

```
int y = 2;

fazAlgo(&y);
```

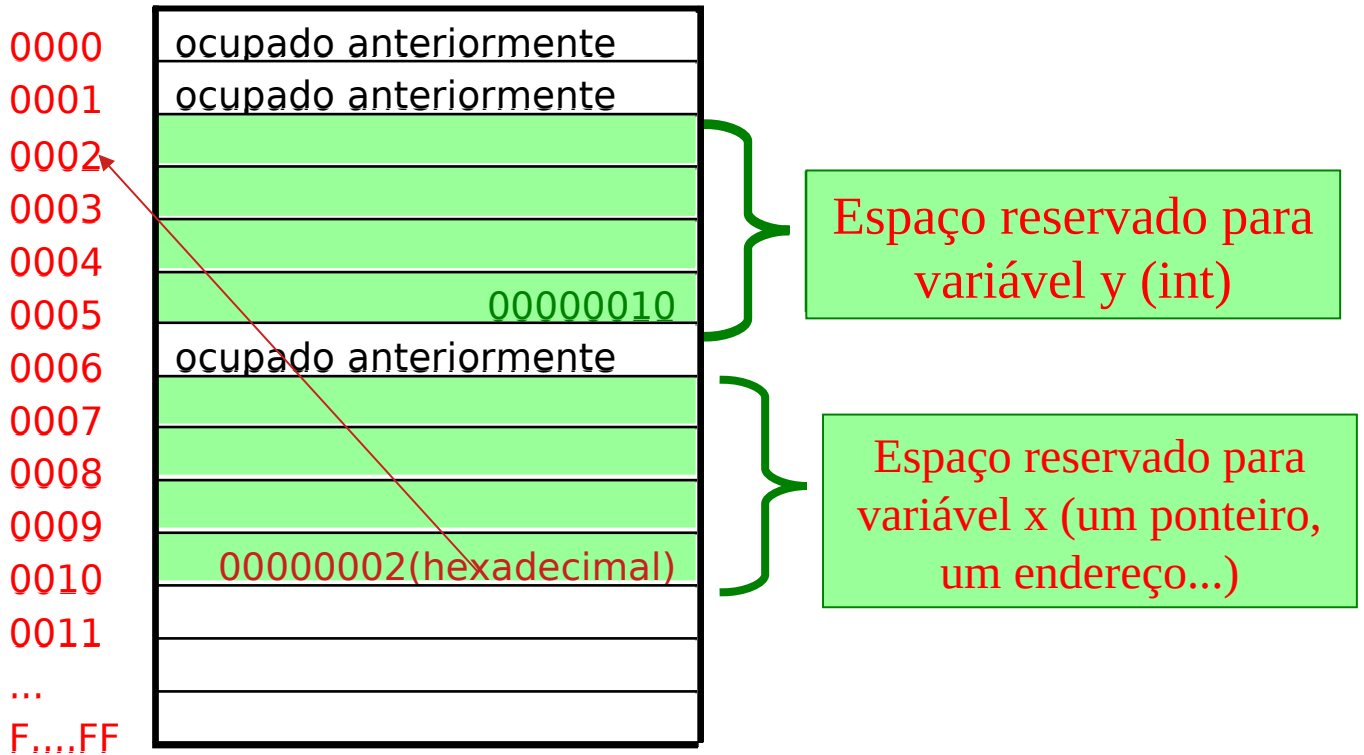


Como simular isso em C?

- Passando o endereço (ponteiro) da variável a ser alterada: o endereço é COPIADO (passagem por valor, afinal), mas aponta para o mesmo espaço em memória que contém a variável que você quer alterar!!!)

```
void fazAlgo (int* x)
{
        (…)
        (*x)++;
        (…)
}
```

```
int y = 2;

fazAlgo(&y);
```

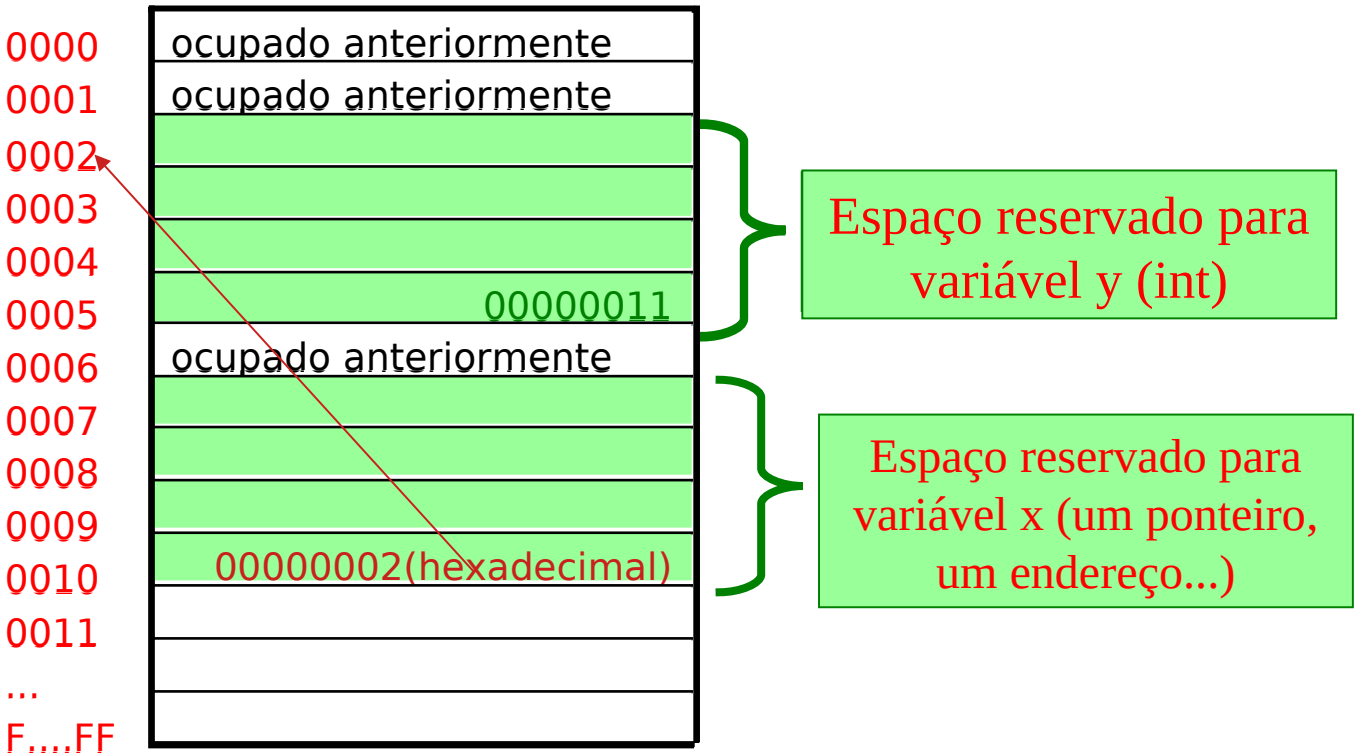


Como simular isso em C?

- Passando o endereço (ponteiro) da variável a ser alterada: o endereço é COPIADO (passagem por valor, afinal), mas aponta para o mesmo espaço em memória que contém a variável que você quer alterar!!!)

```
void fazAlgo (int* x)
{
        (…)
        (*x)++;
        (…)
}
```

```
int y = 2;

fazAlgo(&y);
```



Como simular isso em C?

- Passando o endereço (ponteiro) da variável a ser alterada: o endereço é COPIADO (passagem por valor, afinal), mas aponta para o mesmo espaço em memória que contém a variável que você quer alterar!!!)

```
void fazAlgo (int* x)
{
        (…)
        (*x)++;
        (…)
}
```

```
int y = 2;

fazAlgo(&y);
```

Outro exemplo clássico: troca

```
void troca(int* x, int* y)
{
   int temp = *x;
   *x = *y;
   *y = temp;
}
```

```
//Chamada:

int x, y;
...
troca(&x,&y);
```

Como escrever uma função “troca” que funcione em C?

Uso de ponteiros!

Variáveis que armazenam o endereço de uma variável

O OPERADOR unário “&” fornece o endereço de uma variável

O OPERADOR unário “*” acessa o objeto que um apontador (endereço) aponta

Passagem de parâmetros

Quem quiser rever essas explicações:

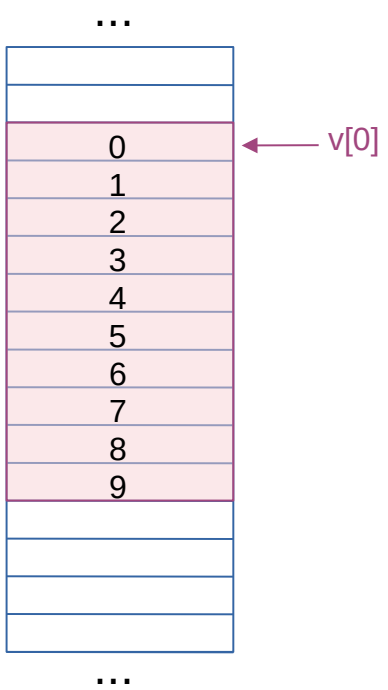
https://eaulas.usp.br/portal/video?idItem=30404

Exemplo: o que essa função (em C) faz?

```c
#include <stdio.h>

void funcaoMisteriosa(int n, int* array){
  for (int i = 0; i < n; i++){
    (*array)++;
    array++;
  }
}

int main(void){
  int v[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  funcaoMisteriosa(10, v);
  for (int i = 0; i < 10; i++)
    printf("%d ", v[i]);
  printf("\n");
}
```

Exemplo: o que essa função (em C) faz?

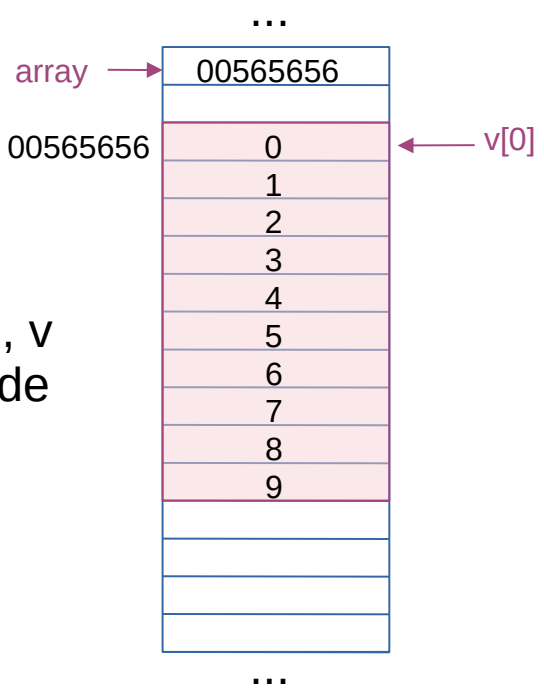
```c
#include <stdio.h>

void funcaoMisteriosa(int n, int* array){
  for (int i = 0; i < n; i++){
    (*array)++;
    array++;
  }
}

int main(void){
  int v[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  funcaoMisteriosa(10, v);
  for (int i = 0; i < 10; i++)
    printf("%d ", v[i]);
  printf("\n");
}
```



Exemplo: o que essa função (em C) faz?

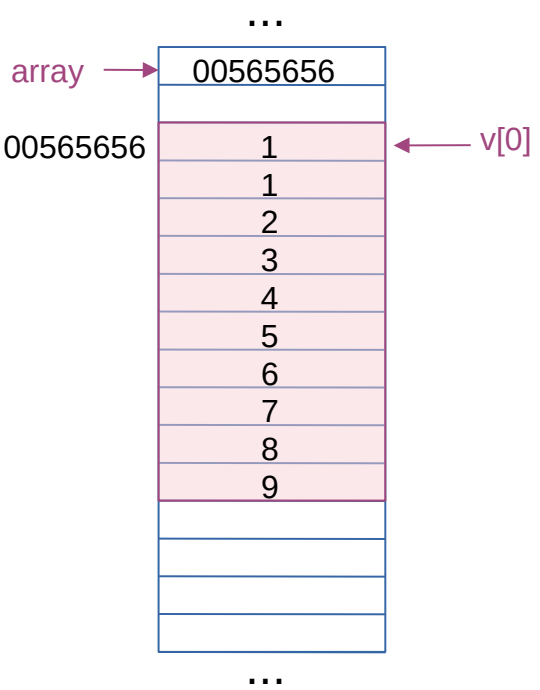
```c
#include <stdio.h>

void funcaoMisteriosa(int n, int* array){
  for (int i = 0; i < n; i++){
    (*array)++;
    array++;
  }
}

int main(void){
  int v[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  funcaoMisteriosa(10, v);
  for (int i = 0; i < 10; i++)
    printf("%d ", v[i]);
  printf("\n");
}
```

Ou seja, mesmo que alocado estaticamente, v também contém o endereço de memória onde começa o vetor



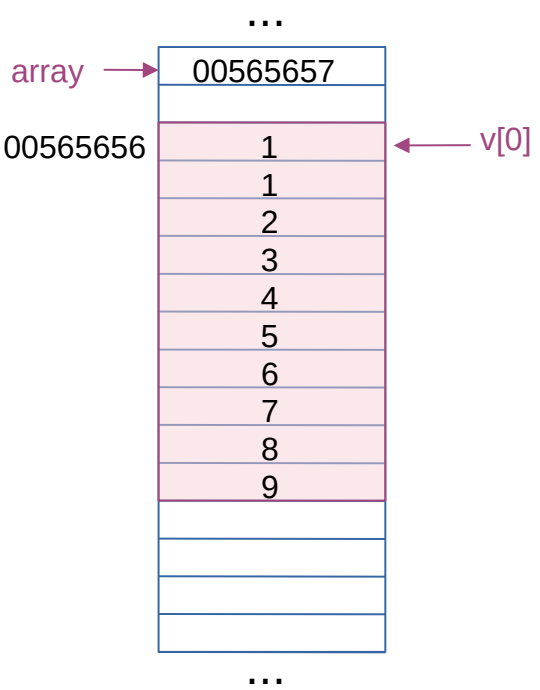
Exemplo: o que essa função (em C) faz?

```c
#include <stdio.h>

void funcaoMisteriosa(int n, int* array){
  for (int i = 0; i < n; i++){
    (*array)++;
    array++;
  }
}

int main(void){
  int v[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  funcaoMisteriosa(10, v);
  for (int i = 0; i < 10; i++)
    printf("%d ", v[i]);
  printf("\n");
}
```



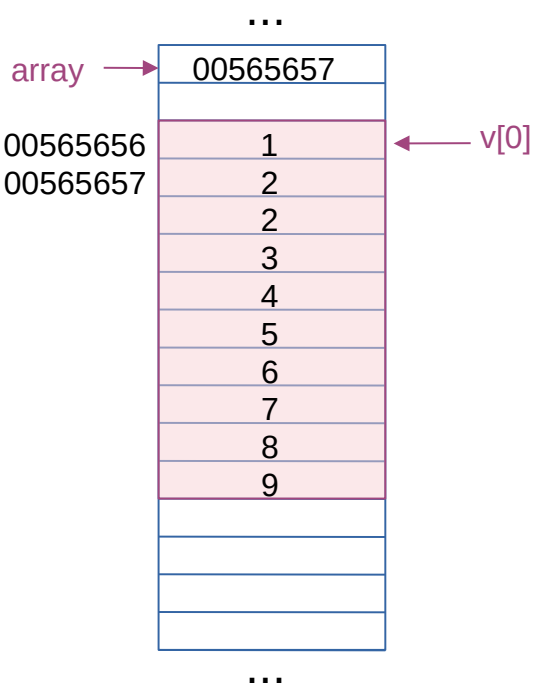
Exemplo: o que essa função (em C) faz?

```c
#include <stdio.h>

void funcaoMisteriosa(int n, int* array){
  for (int i = 0; i < n; i++){
    (*array)++;
    array++;
  }
}

int main(void){
  int v[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  funcaoMisteriosa(10, v);
  for (int i = 0; i < 10; i++)
    printf("%d ", v[i]);
  printf("\n");
}
```



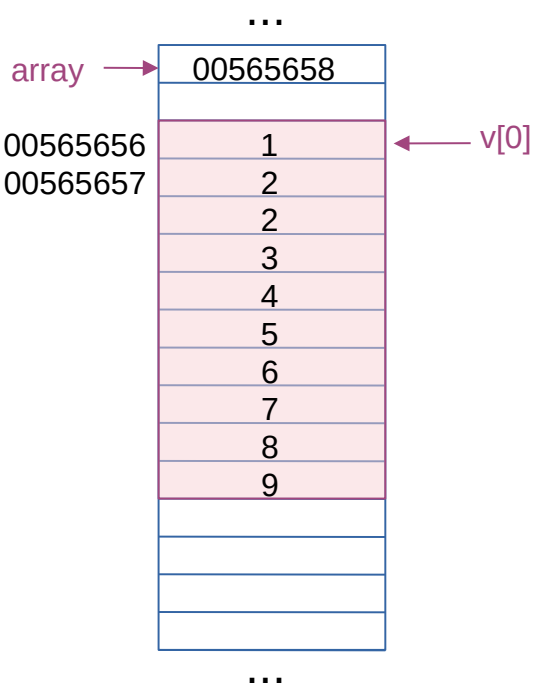
Exemplo: o que essa função (em C) faz?

```c
#include <stdio.h>

void funcaoMisteriosa(int n, int* array){
  for (int i = 0; i < n; i++){
    (*array)++;
    array++;
  }
}

int main(void){
  int v[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  funcaoMisteriosa(10, v);
  for (int i = 0; i < 10; i++)
    printf("%d ", v[i]);
  printf("\n");
}
```



Exemplo: o que essa função (em C) faz?

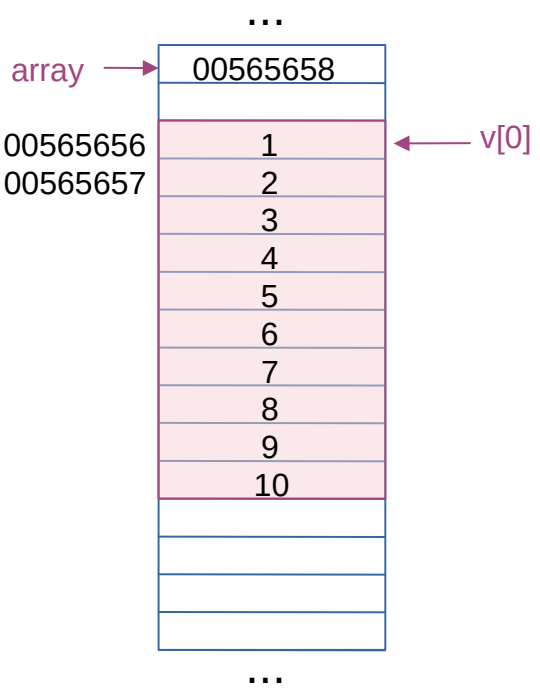
```c
#include <stdio.h>

void funcaoMisteriosa(int n, int* array){
  for (int i = 0; i < n; i++){
    (*array)++;
    array++;
  }
}

int main(void){
  int v[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  funcaoMisteriosa(10, v);
  for (int i = 0; i < 10; i++)
    printf("%d ", v[i]);
  printf("\n");
}
```



Exemplo: o que essa função (em C) faz?

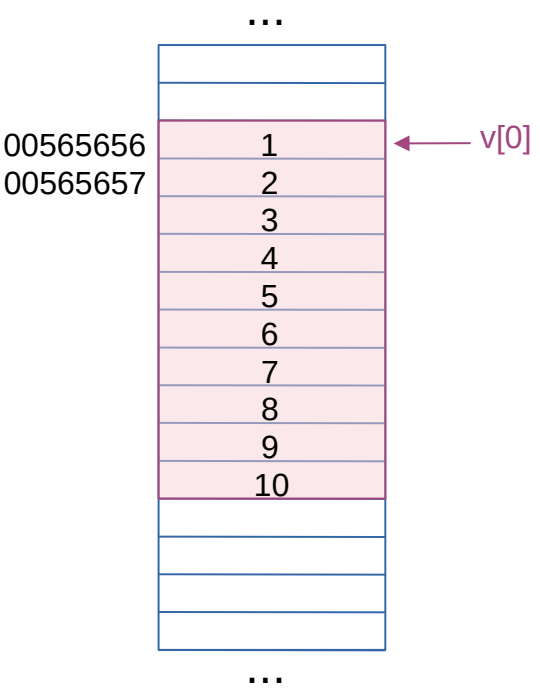
```c
#include <stdio.h>

void funcaoMisteriosa(int n, int* array){
  for (int i = 0; i < n; i++){
    (*array)++;
    array++;
  }
}

int main(void){
  int v[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  funcaoMisteriosa(10, v);
  for (int i = 0; i < 10; i++)
    printf("%d ", v[i]);
  printf("\n");
}
```

n vezes...



Exemplo: o que essa função (em C) faz?

```c
#include <stdio.h>

void funcaoMisteriosa(int n, int* array){
  for (int i = 0; i < n; i++){
    (*array)++;
    array++;
  }
}

int main(void){
  int v[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  funcaoMisteriosa(10, v);
  for (int i = 0; i < 10; i++)
    printf("%d ", v[i]);
  printf("\n");
}
```



Exemplo: o que essa função (em C) faz?

```c
#include <stdio.h>

void funcaoMisteriosa(int n, int* array){
  for (int i = 0; i < n; i++){
    (*array)++;
    array++;
  }
}

int main(void){
  int v[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  funcaoMisteriosa(10, v);
  for (int i = 0; i < 10; i++)
    printf("%d ", v[i]);
  printf("\n");
}
```

Incrementa o **conteúdo do que está sendo apontado** por array (afeta v)

Incrementa o **conteúdo da variável** array, ou seja, vai apontar para a próxima posição do array (**aritmética de ponteiros**)

Passagem por valor x passagem por referência

... Fecha parêntesis !

Voltando ao nosso Grafo

```c
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100
#define AN -1  /* aresta nula, ou seja, valor que representa ausencia de aresta */

typedef int Peso;
typedef struct  {
  Peso mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

Seja:

`Grafo g;`

Como você acessa o campo numVertices dessa estrutura?

Voltando ao nosso Grafo

```c
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100
#define AN -1  /* aresta nula, ou seja, valor que representa ausencia de aresta */

typedef int Peso;
typedef struct  {
  Peso mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

Seja:

`Grafo g;`

Como você acessa o campo numVertices dessa estrutura?

g.numVertices

Voltando ao nosso Grafo

```c
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100
#define AN -1  /* aresta nula, ou seja, valor que representa ausencia de aresta */

typedef int Peso;
typedef struct  {
  Peso mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

Seja:

`Grafo* g;`

Como você acessa o campo numVertices dessa estrutura?

Voltando ao nosso Grafo

```c
#include <stdbool.h>   /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100
#define AN -1  /* aresta nula, ou seja, valor que representa ausencia de aresta */

typedef int Peso;
typedef struct  {
  Peso mat[MAXNUMVERTICES][MAXNUMVERTICES];
  int numVertices;
  int numArestas;
} Grafo;
```

Seja:

`Grafo* g;`

Como você acessa o campo numVertices dessa estrutura?

g->numVertices

Arquivo grafo_matrizadj.c

```c
#include <stdio.h>
#include "grafo_matrizadj.h"

/*
  InicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vertices
  Vertices vao de 1 a nv.
  Preenche as celulas com AN (representando ausencia de aresta)
  Retorna true se inicializou com sucesso e false caso contrario
*/
bool inicializaGrafo(Grafo* grafo, int nv)
```

Como você implementaria?

Arquivo grafo_matrizadj.c

```c
#include <stdio.h>
#include "grafo_matrizadj.h"

/*
  InicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vertices
  Vertices vao de 1 a nv.
  Preenche as celulas com AN (representando ausencia de aresta)
  Retorna true se inicializou com sucesso e false caso contrario
*/
bool inicializaGrafo(Grafo* grafo, int nv)
{









  grafo->numVertices = nv;
  grafo->numArestas = 0;
  for ( i = 0; i <  grafo->numVertices; i++)
    { for ( j = 0; j <  grafo->numVertices; j ++)
        grafo->mat[i][j] = AN;
    }
  return true;
}
```

Arquivo grafo_matrizadj.c

```c
#include <stdio.h>
#include "grafo_matrizadj.h"

/*
  InicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vertices
  Vertices vao de 1 a nv.
  Preenche as celulas com AN (representando ausencia de aresta)
  Retorna true se inicializou com sucesso e false caso contrario
*/
bool inicializaGrafo(Grafo* grafo, int nv)
{ int i , j ;
  if (nv > MAXNUMVERTICES) {
    fprintf(stderr, "ERRO na chamada de inicializaGrafo: Numero de vertices maior \
                                                que o maximo permitido de %d.\n", MAXNUMVERTICES);
    return false;
  }
  if (nv <= 0) {
    fprintf(stderr, "ERRO na chamada de inicializaGrafo: Numero de vertices deve ser positivo.\n");
    return false;
  }
  grafo->numVertices = nv;
  grafo->numArestas = 0;
  for ( i = 0; i <  grafo->numVertices; i++)
    { for ( j = 0; j <  grafo->numVertices; j ++)
        grafo->mat[i][j] = AN;
    }
  return true;
}
```

Arquivo grafo_matrizadj.c

```c
#include <stdio.h>
#include "grafo_matrizadj.h"

/*
  InicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vertices
  Vertices vao de 1 a nv.
  Preenche as celulas com AN (representando ausencia de aresta)
  Retorna true se inicializou com sucesso e false caso contrario
*/
bool inicializaGrafo(Grafo* grafo, int nv)
{ int i , j ;
  if (nv > MAXNUMVERTICES) {
    fprintf(stderr, "ERRO na chamada de inicializaGrafo: Numero de vertices maior \
                                                que o maximo permitido de %d.\n", MAXNUMVERTICES);
    return false;
  }
  if (nv <= 0) {
    fprintf(stderr, "ERRO na chamada de inicializaGrafo: Numero de vertices deve ser positivo.\n");
    return false;
  }
  grafo->numVertices = nv;
  grafo->numArestas = 0;
  for ( i = 0; i <  grafo->numVertices; i++)
    { for ( j = 0; j <  grafo->numVertices; j ++)
        grafo->mat[i][j] = AN;
    }
  return true;
}
```

Pausa para dicas de programação (mensagens de erro)

- Mensagens de erro devem ser claras:
  - Saber qual é o problema
  - Saber onde está o problema (função)
  - Ser facilmente detectada
- Podemos separar a saída padrão (normal do programa – por default a tela) da saída de erro padrão (por default a tela)
  - stdout (saída padrão), stderr (saída de erro)
  - **printf**(…), e **fprintf**(stdout, ….) imprime na saída padrão
  - **fprintf**(stderr, ….) imprime na saída de erro
  - Podem ser redirecionadas para arquivos, para a entrada de outros de programas ou para um “buraco negro”
  - Separar as duas saídas facilita identificar erros ou outras mensagens, principalmente aquelas que não são emitidas logo antes de um “exit”.

Arquivo grafo_matrizadj.c

```c
#include <stdio.h>
#include "grafo_matrizadj.h"

/*
  InicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vertices
  Vertices vao de 1 a nv.
  Preenche as celulas com AN (representando ausencia de aresta)
  Retorna true se inicializou com sucesso e false caso contrario
*/
bool inicializaGrafo(Grafo* grafo, int nv)
{ int i , j ;
  if (nv > MAXNUMVERTICES) {
    fprintf(stderr, "ERRO na chamada de inicializaGrafo: Numero de vertices maior \
                                                que o maximo permitido de %d.\n", MAXNUMVERTICES);
    return false;
  }
  if (nv <= 0) {
    fprintf(stderr, "ERRO na chamada de inicializaGrafo: Numero de vertices deve ser positivo.\n");
    return false;
  }
  grafo->numVertices = nv;
  grafo->numArestas = 0;
  for ( i = 0; i <  grafo->numVertices; i++)
    { for ( j = 0; j <  grafo->numVertices; j ++)
        grafo->mat[i][j] = AN;
    }
  return true;
}
```

Arquivo grafo_matrizadj.c

```c
#include <stdio.h>
#include "grafo_matrizadj.h"

/*
  InicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vertices
  Vertices vao de 1 a nv.
  Preenche as celulas com AN (representando ausencia de aresta)
  Retorna true se inicializou com sucesso e false caso contrario
*/
bool inicializaGrafo(Grafo* grafo, int nv)
{ int i , j ;
  if (nv > MAXNUMVERTICES) {
    fprintf(stderr, "ERRO na chamada de inicializaGrafo: Numero de vertices maior \
                                                que o maximo permitido de %d.\n", MAXNUMVERTICES);
    return false;
  }
  if (nv <= 0) {
    fprintf(stderr, "ERRO na chamada de inicializaGrafo: Numero de vertices deve ser positivo.\n");
    return false;
  }
  grafo->numVertices = nv;
  grafo->numArestas = 0;
  for ( i = 0; i <  grafo->numVertices; i++)
    { for ( j = 0; j <  grafo->numVertices; j ++)
        grafo->mat[i][j] = AN;
    }
  return true;
}
```

Chamada da função:

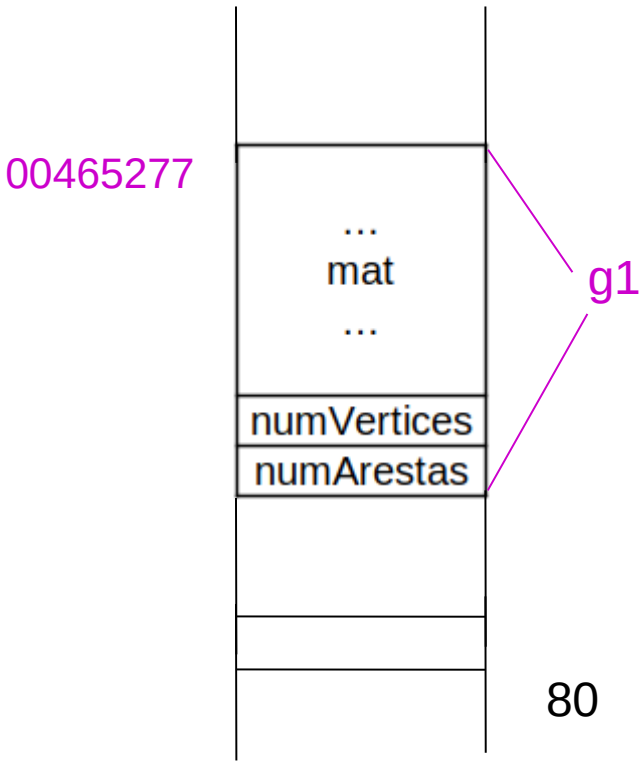
Grafo g;
inicializaGrafo(&g, 10);

Arquivo testa_grafo_matrizadj.c

```c
#include "grafo_matrizadj.h"


int main()
{
  Grafo g1;
  int numVertices;

  inicializaGrafo(&g1, 10);

  //imprimeGrafo(&g1);

  return 0;
}
```

Variável tipo Grafo (struct)

Endereço (referência) da struct armazenada em g1



Arquivo testa_grafo.c

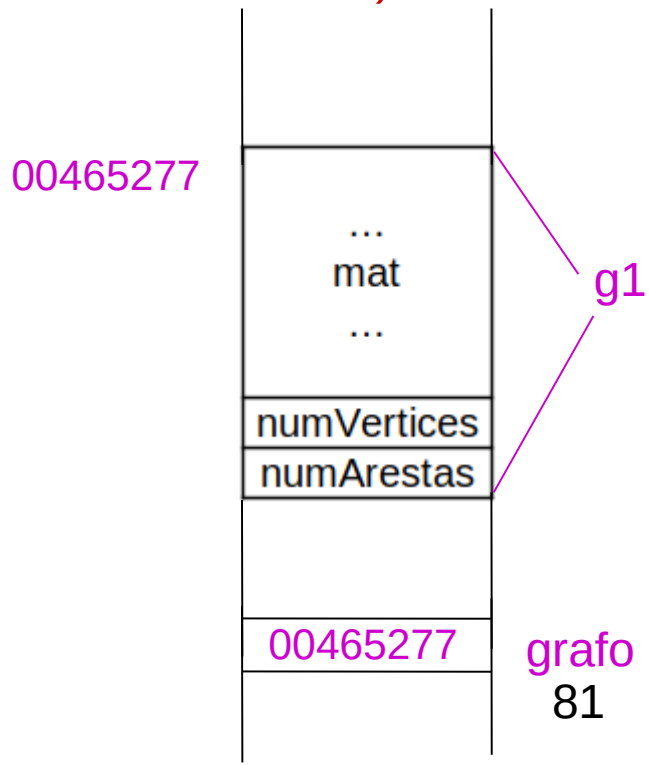
```c
#include "grafo_matrizadj.h"


int main()
{
  Grafo g1;
  int numVertices;

  inicializaGrafo(&g1, 10);

  //imprimeGrafo(&g1);

  return 0;
}
```

```c
bool inicializaGrafo(Grafo* grafo, int nv)
```

grafo é um ponteiro, uma variável que contém um endereço de memória (no caso o endereço de uma struct Grafo)

Variável tipo Grafo (struct)

Endereço (referência) da struct armazenada em g1



Arquivo testa_grafo.c

```c
#include "grafo_matrizadj.h"
#include <stdio.h>

int main()
{
  Grafo g1;
  int numVertices;

  //inicializaGrafo(&g1, 10);

  do {
   printf("Digite o número de vértices do grafo\n");
   scanf("%d", &numVertices);
  } while (!inicializaGrafo(&g1, numVertices));


  //imprimeGrafo(&g1);

  return 0;
}
```

Complexidades
(considerando um grafo de v vértices e a arestas)

	Matriz de adj.	?
inicializaGrafo		
imprimeGrafo		
insereAresta		
existeAresta		
removeAresta		
listaAdjVazia		
proxListaAdj		
liberaGrafo		

Complexidades
(considerando um grafo de v vértices e a arestas)

	Matriz de adj.	?
inicializaGrafo	$O(v^2)$	
imprimeGrafo	$O(v^2)$	
insereAresta		
existeAresta		
removeAresta		
listaAdjVazia		
proxListaAdj		
liberaGrafo		

Complexidades
(considerando um grafo de v vértices e a arestas)

	Matriz de adj.	?
inicializaGrafo	$O(v^2)$	
imprimeGrafo	$O(v^2)$	
insereAresta		
existeAresta		
removeAresta		
listaAdjVazia		
proxListaAdj		
liberaGrafo		

Tentem implementar as demais

Para compilar tudo (por enquanto...)

```
$ gcc -c grafo_matrizadj.c
$ gcc -c testa_grafo.c
$ gcc -o testa_grafo.exe grafo_matrizadj.o testa_grafo.o
```

Referências

ZIVIANI, N. Projetos de Algoritmos - com implementações em Pascal e C. 3ª ed. revista e ampliada Cengage Learning, 2011. (**cap 7**)