UFABC

Vetores e Matrizes

Prof. Fabrício Olivetti de França
Charles Henrique

Vetores Estáticos

Um vetor em C é declarado como:

tipo nome[TAMANHO];

UFABC

Vetores Estáticos

```
/* vetor de nome v1 com 100 elementos do tipo inteiro */
int v1[100];

/* vetor de nome v1 com 32 elementos do tipo char */
char palavra[32];
```

UFABC

Vetores Estáticos

Os vetores em C são chamados de **array** que é um container de elementos **de mesmo tipo** com tamanho fixo.

Uma vez criado, não se pode alterar seu tamanho.

UFABC

Endereçamento de Memória

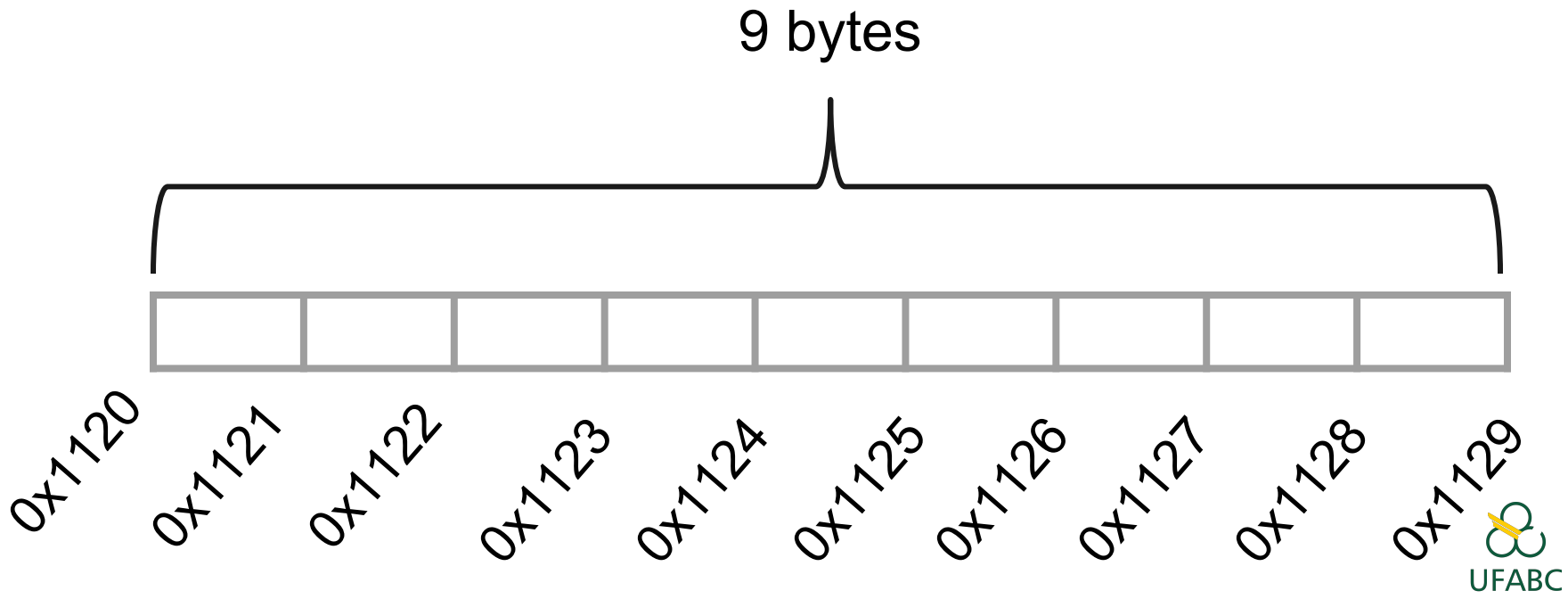
Quando uma array é criada o programa reserva (aloca) o espaço total da memória necessário definido por:

tamanho * sizeof(tipo)

UFABC

Endereçamento de Memória

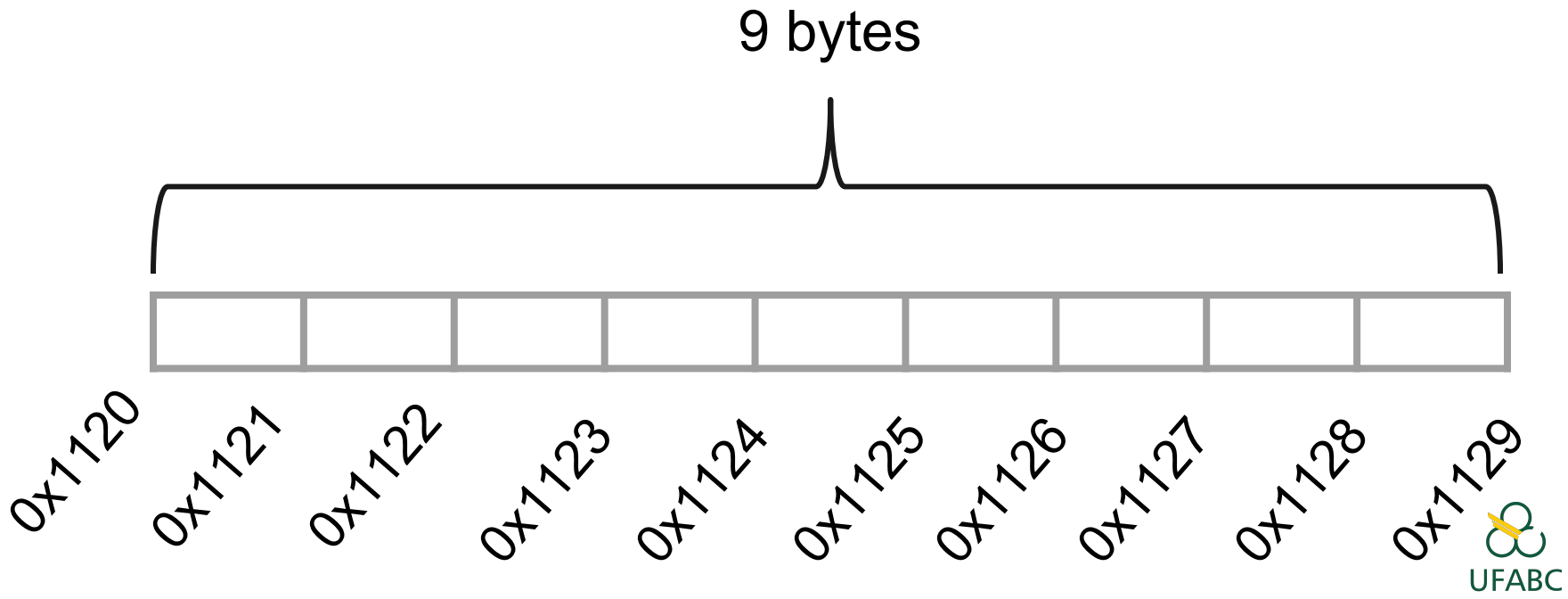
A variável receberá como valor o **endereço da memória** que inicia o espaço reservado da array:



UFABC

Endereçamento de Memória

Endereços geralmente são representados como números hexadecimais. Cada unidade representando um byte.



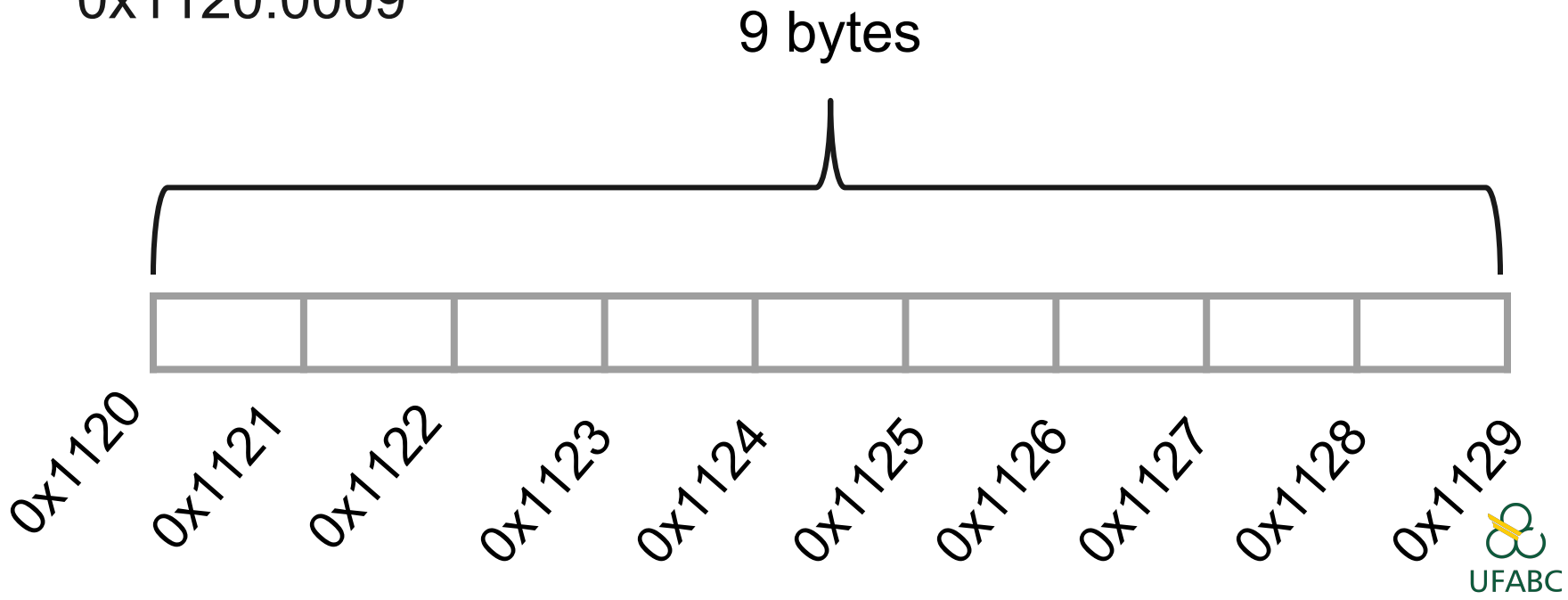
UFABC

Endereçamento de Memória

O espaço reservado é representado por:

INICIO:OFFSET (tamanho)

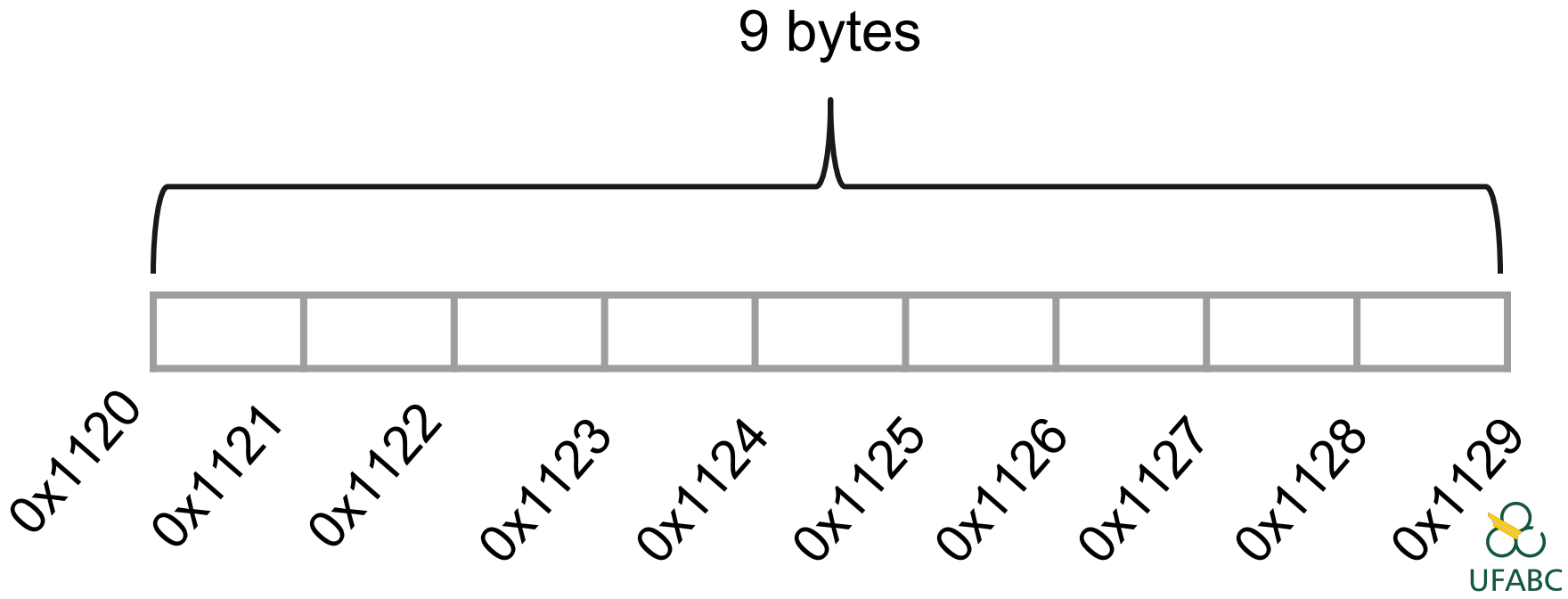
0x1120:0009



UFABC

Endereçamento de Memória

Nesse exemplo a array receberá o valor 0x1120 representando o início da array.



UFABC

Endereçamento de Memória

```
int x[100];
printf("%d", x);
```

x contém o endereço de memória do primeiro elemento.

UFABC

Endereçamento de Memória

Seguindo o exemplo anterior em que **x = 0x1120**, quando acessamos um elemento da array:

X[4] = 10;

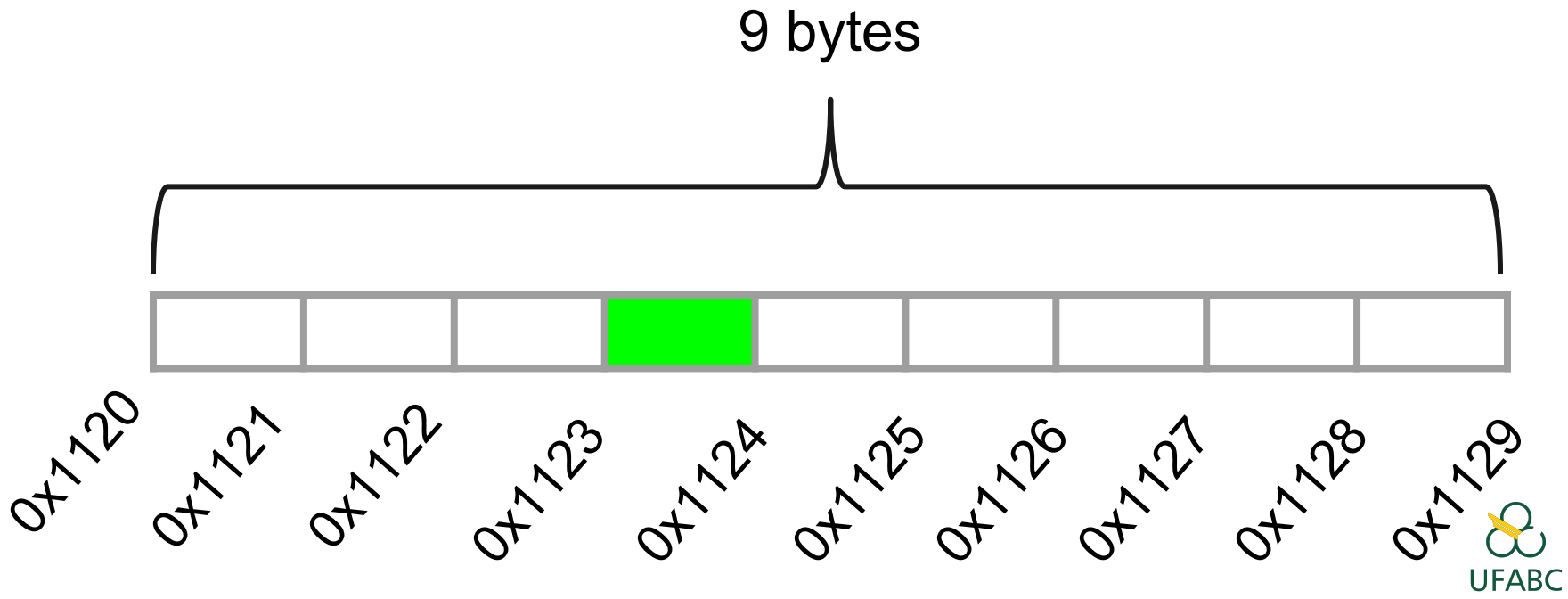
O compilador realizar o cálculo **0x1120 + sizeof(tipo)*4** e retorna **sizeof(tipo) bytes** contidos na memória.

UFABC

Endereçamento de Memória

char x[9];

X[3]; ← 0x1120 + 3*1 = 0x1123

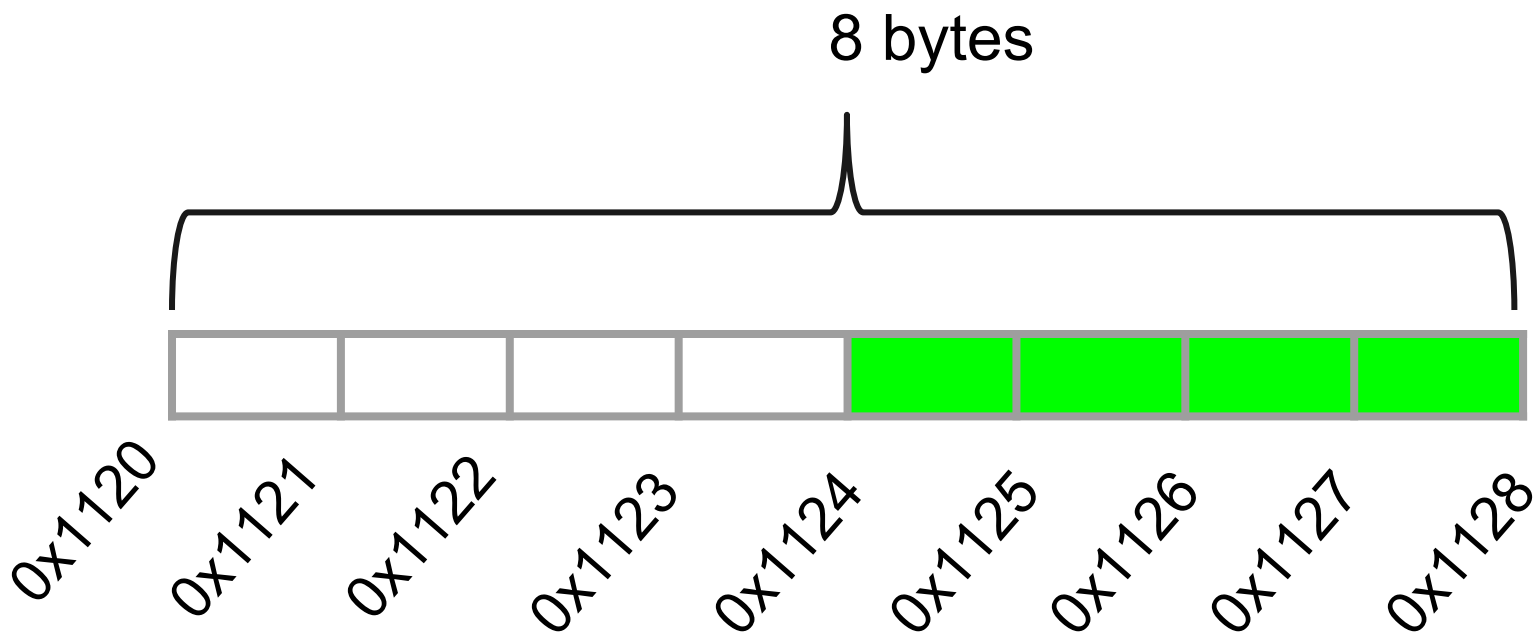


UFABC

Endereçamento de Memória

float x[2];

X[1]; ← 0x1120 + 1*4 = 0x1124



UFABC

Endereçamento de Memória

Por essa razão o primeiro elemento de uma array é o **0**.

UFABC

Endereçamento de Memória

Uma outra forma de acessar um elemento de uma array é calculando o endereço diretamente:

```
int x[10];
printf(“O elemento de índice 3 é: %d\n”, *(x+3));
```

UFABC

Endereçamento de Memória

Quando calculamos **x+3** o compilador automaticamente traduz para **x+3*sizeof(int)**, e retorna o endereço de memória correspondente.

O operador unário * indica que queremos o **conteúdo** daquele endereço de memória.

UFABC

Arrays como Parâmetros

Para receber uma array como parâmetro fazemos:

```
tipo funcao( tipo array[ ] );
```

UFABC

Arrays como Parâmetros

Como o tipo array não carrega consigo seu tamanho, costuma-se fazer:

```
tipo funcao( tipo array[ ], int tamanho );
```

UFABC

Arrays como Parâmetros

```
float media( int dados[ ], int tamanho )
{
    int i;
    float media = 0.0;
    for( i = 0; i<tamanho; i++ ) {
        media += dados[i];
    }
    return media/tamanho;
}
```

UFABC

Arrays como Parâmetros

As arrays, diferente dos tipos básicos, são passadas para as funções como **referência**, ou seja, a função recebe o endereço de memória dela.

Por causa disso, qualquer alteração feita nela será alterada na variável original.

UFABC

Arrays como Parâmetros

```
void muda_array( int dados[ ], int tamanho )
{
    dados[2] = 3;
}
```

UFABC

Arrays como Parâmetros

```
int main ( )
{
    int meus_dados[4] = {1, 1, 1, 1};
    muda_array(meus_dados, 4);
    printf("%d\n", meus_dados[2]);
    return 0;
}
```

UFABC

Exercício 01

Implemente a função:

```
int produto_interno(int x[ ], int y[], int n)
{


}
```

UFABC

Exercício 01

Qual o total de memória utilizada por sua função?

Quantas instruções são executadas?

UFABC

Arrays multi-dimensionais

Similarmente, podemos definir uma array com mais do que uma dimensão:

```
/* matriz 5x5 */
int A[5][5];

/* 10 documentos contendo 20 palavras de até 12 caracteres */
char corpus[10][20][12];
```

UFABC

Endereçamento de Memória

Notem que arrays multi-dimensionais são, na verdade, arrays unidimensionais em que o compilador calcula automaticamente a posição da coordenada.

UFABC

Array Estática

int array [N][M];

Ocupa um segmento contínuo de N*M*sizeof(int) bytes.

Isso nos permite fazer *(&(array[0][0]) + i*M + j) para acessar o elemento (i,j) da array.

UFABC

Array Estática

Esse tipo de array pode ser passada para uma função cuja declaração é:

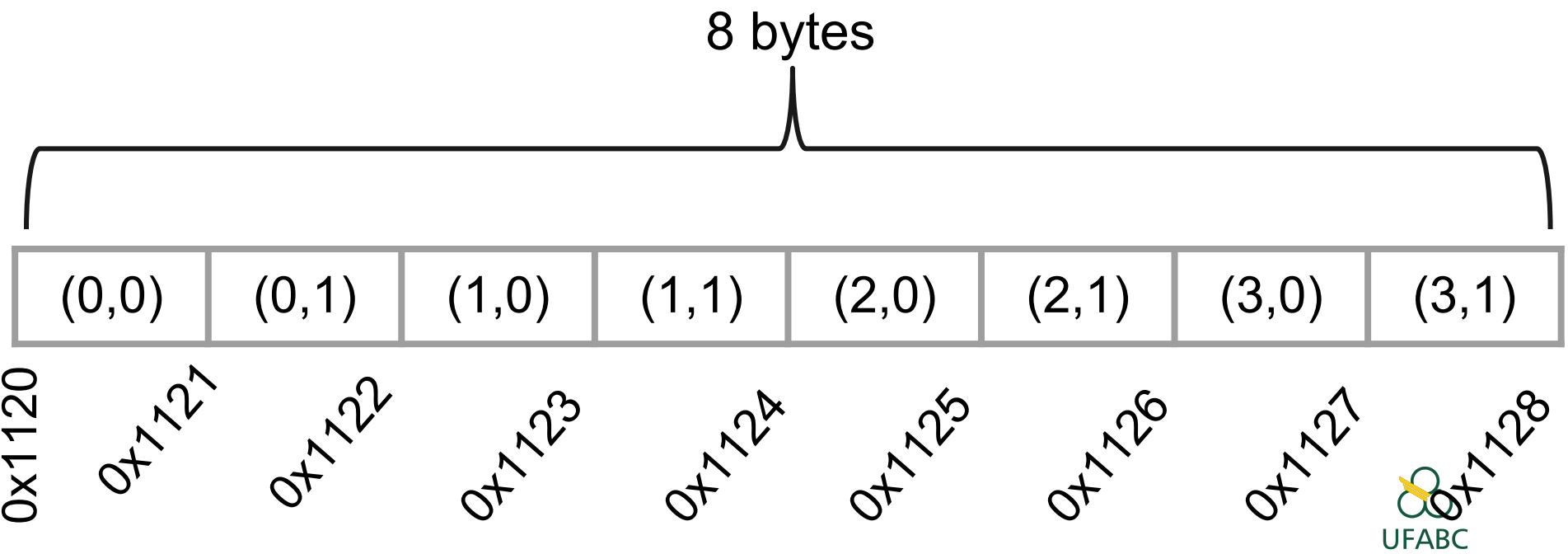
funcao(int array[ ][M]);

Ou seja, devemos especificar as dimensões da array na declaração. Isso é necessário para que o compilador transforme a indexação em aritmética de ponteiros.

UFABC

Endereçamento de Memória

char x[2][4]; ← 8 bytes

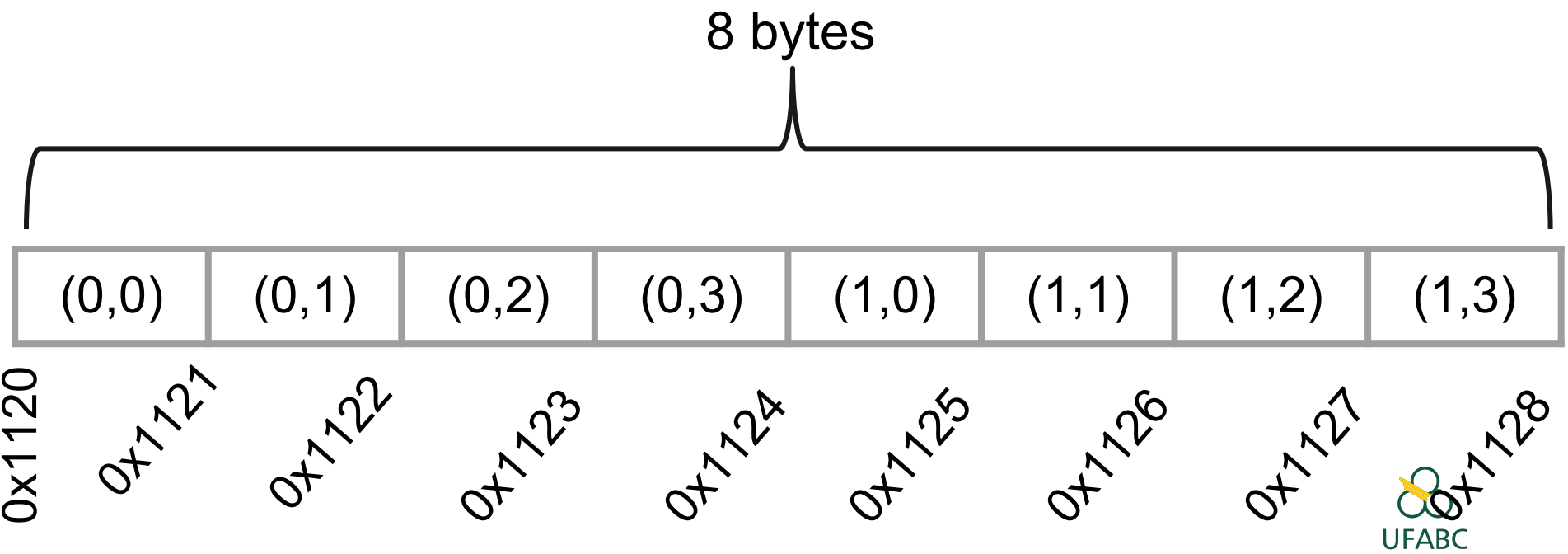


UFABC

Endereçamento de Memória

x[i][j] = *(x + i*COLUNAS + j)

x[1][0] = *(x + 1*4 + 0) = *(0x1120 + 4)



UFABC

Arrays como Parâmetros

```
/* média de uma array com 2 colunas */
float media (int dados[ ][2], int m)
{
...
    for( i = 0; i<m; i++ ) {
        for( j=0; j<2; j++ ) {
            media += dados[i][j];
        }
    }
...
```

UFABC

Exercício 02

Implemente a função:

```
int distancia_euclidiana(int x[ ][2], int n, int i)
{



}
```

Que retorna a média das distâncias euclidianas de x[i] com todos os outros pontos.

UFABC

Exercício 02

Qual o total de memória utilizada por sua função?

Quantas instruções são executadas?

UFABC

Strings

Em C não existe o tipo string!

Para criar uma string utilizamos uma array de char.

Cada elemento da array corresponde a um caractere da string, o último caractere deve ser o `\0` que tem valor 0.

UFABC

Strings

```
char string[100] = “Ola mundo!”;

printf(“%s\n”, string);
```

UFABC

Strings

```
char string[100];

string[0] =  ‘O’;
string[1] =  ‘l’;
string[2] =  ‘a’;

printf(“%s\n”, string);
```

UFABC

Strings

```
char string[100];

string[0] =  ‘O’;
string[1] =  ‘l’;
string[2] =  ‘a’;
string[3] = ‘\0’;

printf(“%s\n”, string);
```

UFABC

Strings

Em C uma sequência de caracteres, ou string, é definida entre aspas duplas.

Um único caractere é definido por aspas simples.

UFABC

Exercício 03

Implemente a função:

```
int length(char s[ ])
{



}
```

Que retorna o tamanho da string s.

UFABC

Exercício 03

Qual o total de memória utilizada por sua função?

Quantas instruções são executadas?

UFABC