>>> Programação Orientada a Objetos (POO)

... Vetores

Prof: André de Freitas Smaira

[~]$ _

```
ian@aspireE1571:/tmp$ xxd /bin/bash | grep cafe
00045490: ffe9 cafe ffff 662e 0f1f 8400 0000 0000  ......f.........
00052e60: 8628 feff ff48 85c0 0f84 cafe ffff 89c7  .(...H..........
00069b10: 488b 7b08 e8d7 cafe ff48 8943 0848 8b5b  H.{......H.C.H.[
00079b10: e7e8 9af6 ffff a802 0f84 cafe ffff e9aa  ................
0009f2d0: 8b05 cafe 2600 85c0 7426 85ff 488d 051d  ....&...t&..H...
000a0450: 562a 0400 4531 ffe9 cafe ffff 0f1f 4000  V*..E1........@.
000a0740: 0249 89ed 4c89 7c24 08e9 cafe ffff 6690  .I..L.|$......f.
000a86f0: 0f85 cafe ffff 5b31 c05d 415c 415d 415e  ......[1.]A\A]A^
000c91c0: 85d2 0f8e cafe ffff 4183 e41f 4901 c483  ........A...I...
000cafe0: 0800 3b00 0a00 0b00 0c00 0d00 ffff ffff  ..;.............
```

Organizando dados na memória

[1. Fui tapeado! O que escondemos de vocês]$ _

>>> Vamos ver um pouco mais de Arrays

* **Declaração estática**: <tipo> <nome>[tamanho]
  Ex: `int v[100];`
* **Declaração com inicialização**:
  <tipo> <nome>[tamanho] = {e_1, e_2, ..., $e_{tamanho}$}
  Ex: `char s[4] = {'a','b','c','\0'}; //'\0' == NULL!`
  <tipo> <nome>[] = {e_1, e_2, ...}
  Ex: `char s[] = "abc"; // Recomendado: automático!`
* **Acesso**: <nome>[indice]
  Ex: `v[i];`
* **Atribuição**: <nome>[indice] = <valor>
  Ex: `v[85] = 42;`

>>> vetores.cpp

```cpp
#include <iostream>
int main()
{
    int i;
    // "s" guarda o endereço do vetor tamanho de 3 int
    int s[3];

    s[0] = 100; // Acessa o primeiro elemento e atribui 100
    s[1] = 200; // Acessa o segundo  elemento e atribui 200
    s[2] = 300; // Acessa o terceiro elemento e atribui 300
    s[3] = 400; // Acessa o... como assim?!
    // Funciona?! Mas o vetor tem apenas 3 elementos!?
    std::cout << s[0] << " "
              << s[1] << " "
              << s[2] << " "
              << s[3] << syd::endl;
    return 0;
}
```

[2. Agrupando bytes: vetores]$ _

>>> vetores.cpp

* Funciona... mas as vezes não
* **Segmentation Fault**: tentou acessar (**espionar**) memória alheia e o **Kernel matou**
* Pense... um programa **malicioso** (o seu no caso) poderia **ler ou alterar a memória**

>>> vetor_memoria.c

* Por que índices começam em 0?
* Como os vetores são guardados na memória?

>>> vetor_memoria.c

* **Vetor**: inteiros **lado a lado** na memória e guarda o **endereço do primeiro**

```
int v[2] = {0xCA, 0xFE}; // {202, 254}
```

* se v no endereço **0xC0FFEE**:

```
v+0 -> 0xC0FFEE (0xC0FFEE + 0*sizeof(int)) // 202
v+1 -> 0xC0FFF2 (0xC0FFEE + 1*sizeof(int)) // 254
v+2 -> 0xC0FFF6 (0xC0FFEE + 2*sizeof(int)) //  ??
```

* **Acesso direto => super rápido**
* **Memória**

```
0xC0FFEE: 00000000 00000000 00000000 11001010
0xC0FFF2: 00000000 00000000 00000000 11111110
0xC0FFF6: Qualquer coisa (lixo)
```

>>> strings.c

* string está para char assim como vetor de inteiros está para int
* "É proibido utilizar acentos nas strings mas se quiser pode"
  É bom evitar xD, pois são duas posições

>>> strings.c: caracteres especiais

```
/* Este programa demonstra o comportamento do C++ frente *
 * A caracteres especiais (por exemplo: "ê" e "é")       */
#include <iostream>
int main()
{
    int i;
    char sou_vetor[] = "sim... você é!";
    std::cout << sou_vetor << std::endl;  // Printa normal...
    std::cout << int(sizeof(sou_vetor)/sizeof(char))
              << std::endl;
    for (i = 0; i < sizeof(sou_vetor); i++)
        // Aqui ele printa caractere por caractere
        // No entanto caracteres especiais ocupam o dobro
        // do espaço... logo ele interpreta errado o caractere
        std::cout << sou_vetor[i] << " "
                  << std::hex << sou_vetor[i]
                  << std::endl;
    return 0;
}
```

>>> Lendo vetores diretamente da entrada

```
#define N 3
#include <stdio.h>
int main()
{
    int i;
    int d[N-1];
    // Lê da entrada (forma alternativa)...
    for (i=0; i<N; i++)
        std::cin >> d[i];
    // Mostra o que foi lido
    for (i=0; i<N; i++)
         std::cout << d[i] << " ";
    std::cout << std::endl;
    return 0;
}
```

>>> Exemplo I

```cpp
/* Este programa troca a caixa da primeira letra da string*/
#include <iostream>
int main()
{
    int i; char a[200];
    scanf("%[^\n]", a); // lê uma linha do terminal
    std::cout << "A string digitada: " << a << std::endl;
    if (a[0] >= 'a' && a[0] <= 'z') // se for minúsculo
        a[0] = 'A' + (a[0]-'a');
    else
        a[0] = 'a' + (a[0]-'A');
    std::cout << "A string modificada: " << a << std::endl;
    return 0;
}
```

>>> Exemplo II

```cpp
#include <iostream>
#include <string>
int main()
{
    int i; std::string a;
    std::getline(std::cin, a);
    std::cout << "A string digitada: " << a << std::endl;
    if (a[0] >= 'a' && a[0] <= 'z') // se for minúsculo
        a[0] = 'A' + (a[0]-'a');
    else
        a[0] = 'a' + (a[0]-'A');
    std::cout << "A string modificada: " << a << std::endl;
    return 0;
}
```

>>> Exemplo III

```cpp
#include <iostream>

int main()
{
    char s[] = "Hello World";
    std::cout << "\"" << s << "\" tem " << int(sizeof(s))
              << " carac" << std::endl;
    // Substituindo '\0' por outra coisa!!!
    s[sizeof(s)-1] = ' '; //só funciona pq sizeof(char) == 1
    s[sizeof(s)] = ' '; //só funciona pq sizeof(char) == 1

    // Na hora de imprimir ele vai até '\0' (0x00) que
    // finaliza a string.O tamanho do vetor é o mesmo
    // pois sizeof retorna o valor que foi alocado no início
    std::cout << "\"" << s << "\" tem " << int(sizeof(s))
              << " carac" << std::endl;
    std::cout << s[-1] << std::endl;
    return 0;
}
```

>>> Exemplo IV

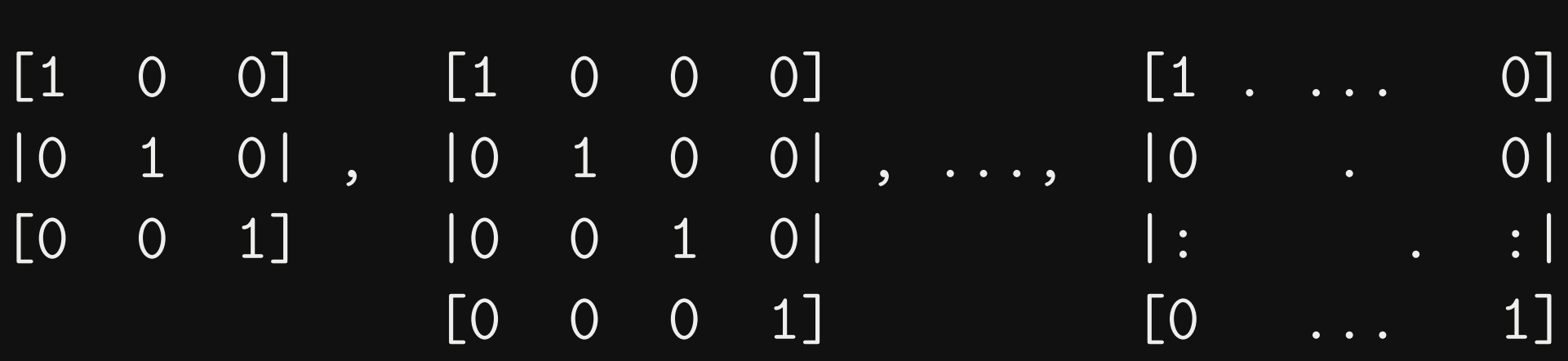
```c
/* Verifica se duas strings são iguais... */
#include <stdio.h>
int main()
{
    int i;
    char a[20], b[20];
    // Lê até a quebra de linha '\n' (Excluindo esta)
    scanf("%[^\n] %[^\n]", a, b);
    for (i = 0; i < strlen(a, b); i++)
        if (a[i] != b[i]) break;
    if (i != 20) printf("São diferentes\n");
    else printf("São iguais\n");
    return 0;
}
```

>>> Exemplo V

```c
/* Verifica se duas strings são iguais... */
#include <stdio.h>
#include <string.h>
int main()
{
    int i;
    char a[20], b[20];
    // Lê até a quebra de linha '\n' (Excluindo esta)
    scanf("%[^\n] %[^\n]", a, b);
    if (strcmp(a, b)) printf("São diferentes\n");
    else printf("São iguais\n");
    return 0;
}
```

>>> Exemplo VI

```cpp
#include <iostream>
#include <string>
int main()
{
    int i;
    std::string a, b;
    // Lê até a quebra de linha '\n' (Excluindo esta)
    std::getline(std::cin, a);
    std::getline(std::cin, b);
    if (a != b) printf("São diferentes\n");
    else printf("São iguais\n");
    return 0;
}
```



$$\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}, \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}, \ldots, \begin{bmatrix}1 & \cdots & 0\\ 0 & \ddots & 0\\ \vdots & & \vdots\\ 0 & \cdots & 1\end{bmatrix}$$

Matrizes... são vetores de... vetores!

>>> Matrizes

* Vetor guardando **endereços de vetores**
  Ex: `int A[n][m];`
* Estamos dizendo ao computador para criar **n vetores com m elementos cada** ($A_{n \times m}$).
* **Acesso**: `A[i][j]` é o elemento a_{ij}.
* **Importantíssimas** na **Computação**, Matemática, Física, etc

>>> Matrizes

O que esse programa escreve?

```cpp
#define N 20
#include <iostream>
int main()
{
    int I[N][N], i=0, j=0;
    // Zerar a matriz...
    for (int i=0; i<N; ++i)
        for (int j=0; j<N; ++j)
            I[i][j] = 0;
    // Preencher a diagonal
    for (int i=0; i<N; ++i) I[i][i] = 1;
    // Acessando que nem gente (o que?)
    std::cout << I[i][j] << std::endl;
    return 0;
}
```

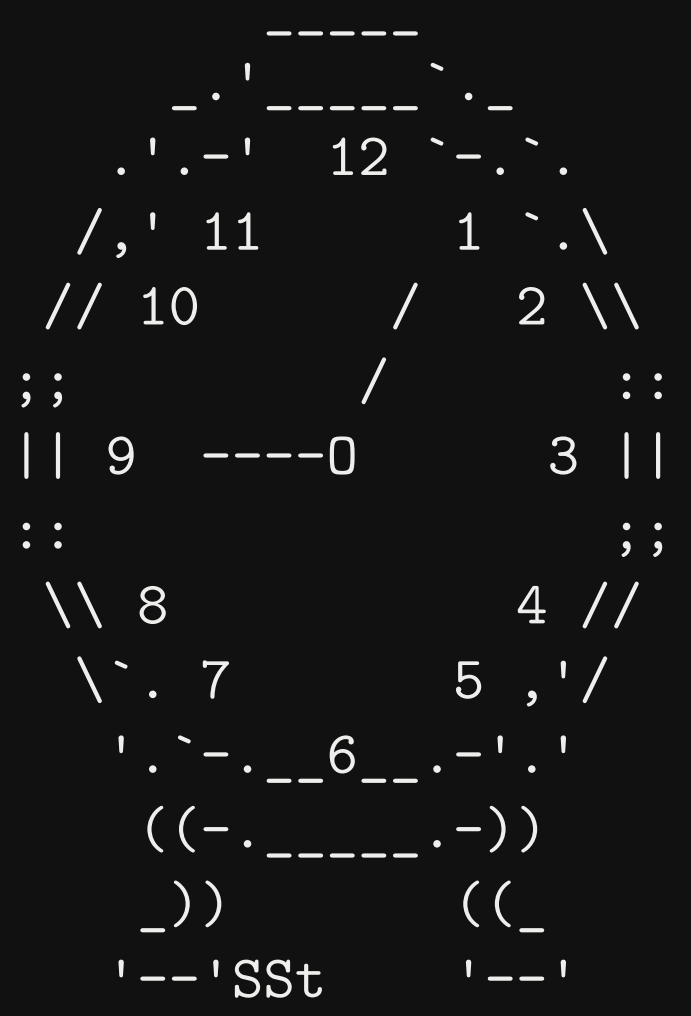
>>> Como a matriz é alocada na memória?

```
#define N 3
...
    /* [0  1  2]     Doce ilusão...
       |3  4  5|   <- Matriz M -> [0 1 2 3 4 5 6 7 8]
       [6  7  8]                                    */
    int M[N][N];
    for(int i=0; i<N; i++)
        for(int j=0; j<N; j++)
             M[i][j] = N*i+j;

    std::cout << "m_{0x0} = " << M[0][0] // 0
              << std::endl;
    std::cout << "m_{0x1} = " << M[0][1] // 1
              << std::endl;
    std::cout << "m_{0x2} = " << M[0][2] // 2
              << std::endl;
    std::cout << "m_{0x3} = " << M[0][3] // 3 - Acesso indevido!
              << std::endl;
    std::cout << "m_{0x4} = " << M[0][4] // 4 - Acesso indevido!
              << std::endl;
    // É só uma abstração... uma matriz é um vetor
    //    indexado de uma forma especial...
```

>>> Exemplos... Ao vivo bixooo!!!

* Como faríamos a **soma/subtração** de duas matrizes?
* Como seria a operação de **transposição** de uma matriz?
* Existem diversas coisas bem mais divertidas para se fazer com matrizes e programação: **Determinantes**, **Autovalores** e **autovetores**, resolver **sistemas lineares**, **projeções** (3D em 2D - exemplo games 3D)



Apontando para as coisas

>>> O que são os ponteiros

* Variável que guarda endereços de memória
* Ponteiro para um tipo é outro tipo
* Passagem de argumentos para funções e structs
* Declaração:

  ```
  tipo *nome;
  ```

* Ex: Ponteiro para inteiro, apontando para nada

  ```
  int *p = NULL;
  ```

* Pode-se ter um ponteiro para um ponteiro também:

  ```
  int **p = NULL;
  ```

>>> Aritmética de ponteiros

* Ao (de)[in]crementar um ponteiro o endereço do ponteiro é (subtraído)[acrescido] no **número de bytes** ocupados pelo tipo. (Lembra de **vetores**? Isso mesmo... o identificador do vetor nada mais é que um **ponteiro**!)
* **Valor apontado** usa-se o **operador unário** `*` (não confundir com a multiplicação, que é binária)
* Se `var` é uma variável e `pon = &var` então `*pon` é `var`

Alocação dinâmica

>>> Alocação dinâmica de memória

* Nem sempre sabemos o **número de elementos** ou depende da execução...
* **malloc**: da biblioteca **stdlib.h** ... **Esquece isso!**
* **Sintaxe**:

```
p = new <tipo>[tamanho];
...
delete[] p;
```

>>> Alocando vetores

```cpp
#include <iostream>
int main() {
    int n;
    double *p = NULL; // inicialmente aponta pra lugar nenhum
    std::cout << "Digite o tamanho do vetor: ";
    std::cin >> n;
    // malloc(x) pega aloca x de espaço, no caso n double(s).
    p = new double[n];
    for (int i=0; i<n; ++i) p[i] = i*i*i;
    for (int i=0; i<n; ++i)
        std::cout << p[i] << std::endl;
    delete[] p; // Libera a memória durante a execução
    p = NULL; // Evitar pontas soltas no seu código.
    return 0;
}
```

>>> Alocando vetores

```cpp
#include <iostream>
int main() {
    int m,n;
    int **M; // Ponteiro para ponteiro
    std::cin >> m >> n;
    M = new int*[m];
    for (int i = 0; i < m; ++i)
        M[i] = new int[n];
}
```

>>> Alocando vetores

```
...
   typedef struct {
      int kills, assists, deaths;
   } score;
   score *noob = new score;
   *(noob).kills   = 3;  // 3 outros noobs xD
   *(noob).assists = 20; // Trabalho em equipe
   noob -> deaths  = 38; // "->" é um atalho para *(noob).deat
   delete noob;
   noob = NULL;
...
```

>>> Referências e Leitura Recomendada

* Aulas do Grupo Maratona IFSC (Ian Giestas Pauli e eu)
* Arrays
  https://www.programiz.com/c-programming/c-arrays
* Alocação de memória
  https://www.ime.usp.br/~pf/algoritmos/aulas/aloca.html
* Ponteiros
  https:
  //www.ime.usp.br/~pf/algoritmos/aulas/pont.html#pointer