Aula 22 – Matrizes (parte 1)

Norton T. Roman & Luciano A. Digiampietri

Arranjos

- Voltemos à necessidade de se dar nomes aos materiais

```
#include <stdio.h>
/* nomes dos materiais */
char nAlvenaria[] = {'A','l','v','e','n','a','r','i','a','\0'};
char nVinil[] = {'V','i','n','i','l','\0'};
char nFibra[] = {'F','i','b','r','a','\0'};
char nPlastico[] = {'P','l','a','s','t','i','c','o','\0'};

int main() {
  printf("Piscina de %s\n",nVinil);
  return 0;
}
```

Arranjos

- Voltemos à necessidade de se dar nomes aos materiais
- Qual o problema?

```
#include <stdio.h>
/* nomes dos materiais */
char nAlvenaria[] = {'A','l','v','e','n','a','r','i','a','\0'};
char nVinil[] = {'V','i','n','i','l','\0'};
char nFibra[] = {'F','i','b','r','a','\0'};
char nPlastico[] = {'P','l','a','s','t','i','c','o','\0'};

int main() {
  printf("Piscina de %s\n",nVinil);
  return 0;
}
```

Arranjos

- Voltemos à necessidade de se dar nomes aos materiais
- Qual o problema?
  - Começou a crescer demais...

```
#include <stdio.h>
/* nomes dos materiais */
char nAlvenaria[] = {'A','l','v','e','n','a','r','i','a','\0'};
char nVinil[] = {'V','i','n','i','l','\0'};
char nFibra[] = {'F','i','b','r','a','\0'};
char nPlastico[] = {'P','l','a','s','t','i','c','o','\0'};

int main() {
  printf("Piscina de %s\n",nVinil);
  return 0;
}
```

Arranjos

- Voltemos à necessidade de se dar nomes aos materiais
- Qual o problema?
  - Começou a crescer demais...
- O que poderíamos fazer?

```
#include <stdio.h>
/* nomes dos materiais */
char nAlvenaria[] = {'A','l','v','e','n','a','r','i','a','\0'};
char nVinil[] = {'V','i','n','i','l','\0'};
char nFibra[] = {'F','i','b','r','a','\0'};
char nPlastico[] = {'P','l','a','s','t','i','c','o','\0'};

int main() {
  printf("Piscina de %s\n",nVinil);
  return 0;
}
```

Arranjos

- Voltemos à necessidade de se dar nomes aos materiais
- Qual o problema?
  - Começou a crescer demais...
- O que poderíamos fazer?
  - Um agrupamento semelhante ao feito com os preços.

```
#include <stdio.h>
/* nomes dos materiais */
char nAlvenaria[] = {'A','l','v','e','n','a','r','i','a','\0'};
char nVinil[] = {'V','i','n','i','l','\0'};
char nFibra[] = {'F','i','b','r','a','\0'};
char nPlastico[] = {'P','l','a','s','t','i','c','o','\0'};

int main() {
  printf("Piscina de %s\n",nVinil);
  return 0;
}
```

Arranjos de Arranjos

- E como seria esse agrupamento?

Arranjos de Arranjos

- E como seria esse agrupamento?
  - Um arranjo de strings

Arranjos de Arranjos

- E como seria esse agrupamento?
  - Um arranjo de strings
  - Um arranjo de arranjos

Arranjos de Arranjos

- E como seria esse agrupamento?
  - Um arranjo de strings
  - Um arranjo de arranjos
  - Uma matriz

Arranjos de Arranjos

- E como seria esse agrupamento?
  - Um arranjo de strings
  - Um arranjo de arranjos
  - Uma matriz
- E como ficaria em C?

Matriz

```
#include <stdio.h>
char nomes[4][10] = {{'A','l','v','e','n','a','r','i','a','\0'},
    {'V','i','n','i','l','\0'}, {'F','i','b','r','a','\0'},
    {'P','l','a','s','t','i','c','o','\0'}};

int main() {

  printf("%s\n",nomes[1]);
  printf("%c\n",nomes[1][2]);

  return 0;
}
```

Matriz

- É necessário que a segunda dimensão tenha tamanho fixo

```
#include <stdio.h>
char nomes[4][10] = {{'A','l','v','e','n','a','r','i','a','\0'},
    {'V','i','n','i','l','\0'}, {'F','i','b','r','a','\0'},
    {'P','l','a','s','t','i','c','o','\0'}};

int main() {

  printf("%s\n",nomes[1]);
  printf("%c\n",nomes[1][2]);

  return 0;
}
```

Matriz

- E como acessamos isso?

```
#include <stdio.h>
char nomes[4][10] = {{'A','l','v','e','n','a','r','i','a','\0'},
    {'V','i','n','i','l','\0'}, {'F','i','b','r','a','\0'},
    {'P','l','a','s','t','i','c','o','\0'}};

int main() {

  printf("%s\n",nomes[1]);
  printf("%c\n",nomes[1][2]);

  return 0;
}
```

Matriz

- E como acessamos isso?
  - O segundo nome:

```
#include <stdio.h>
char nomes[4][10] = {{'A','l','v','e','n','a','r','i','a','\0'},
    {'V','i','n','i','l','\0'}, {'F','i','b','r','a','\0'},
    {'P','l','a','s','t','i','c','o','\0'}};

int main() {

  printf("%s\n",nomes[1]);
  printf("%c\n",nomes[1][2]);

  return 0;
}
```

Matriz

- E como acessamos isso?
  - O segundo nome:
  - A terceira letra do segundo nome:

```
#include <stdio.h>
char nomes[4][10] = {{'A','l','v','e','n','a','r','i','a','\0'},
    {'V','i','n','i','l','\0'}, {'F','i','b','r','a','\0'},
    {'P','l','a','s','t','i','c','o','\0'}};

int main() {

  printf("%s\n",nomes[1]);
  printf("%c\n",nomes[1][2]);

  return 0;
}
```

Matriz

- E como acessamos isso?
  - O segundo nome:
  - A terceira letra do segundo nome:

```
#include <stdio.h>
char nomes[4][10] = {{'A','l','v','e','n','a','r','i','a','\0'},
    {'V','i','n','i','l','\0'}, {'F','i','b','r','a','\0'},
    {'P','l','a','s','t','i','c','o','\0'}};

int main() {

  printf("%s\n",nomes[1]);
  printf("%c\n",nomes[1][2]);

  return 0;
}
```

Saída

```
Vinil
n
```

Matriz

- Notem que *nomes* é do tipo arranjo bidimensional de caracteres

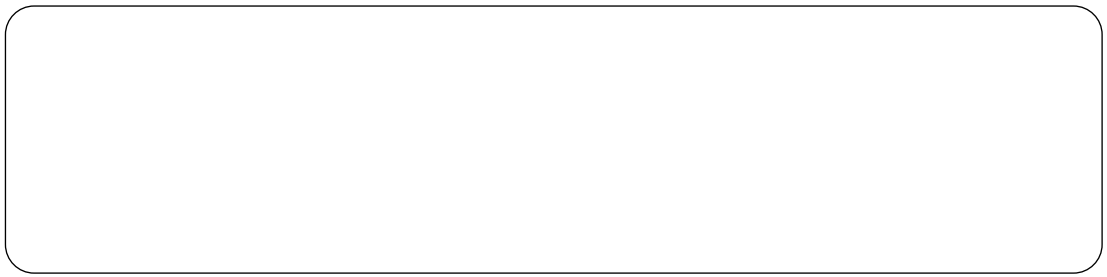
```
#include <stdio.h>
char nomes[4][10] = {{'A','l','v','e','n','a','r','i','a','\0'},
    {'V','i','n','i','l','\0'}, {'F','i','b','r','a','\0'},
    {'P','l','a','s','t','i','c','o','\0'}};

int main() {

  printf("%s\n",nomes[1]);
  printf("%c\n",nomes[1][2]);

  return 0;
}
```

Saída

```
Vinil
n
```

Matriz

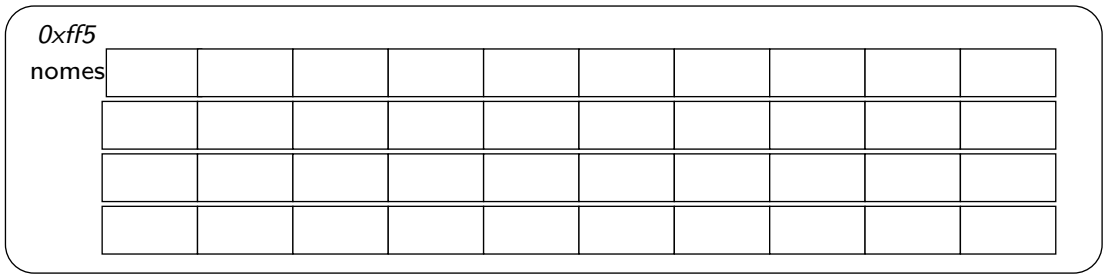
- O que acontece na memória quando declaramos o arranjo?



```
char nomes[4][10] = {{'A','l','v','e','n','a','r','i','a','\0'},
    {'V','i','n','i','l','\0'}, {'F','i','b','r','a','\0'},
    {'P','l','a','s','t','i','c','o','\0'}};
```

Matriz

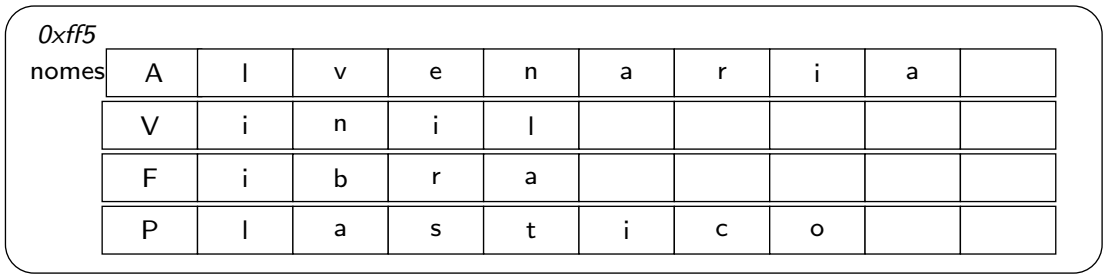
- O que acontece na memória quando declaramos o arranjo?




```
char nomes[4][10] = {{'A','l','v','e','n','a','r','i','a','\0'},
    {'V','i','n','i','l','\0'}, {'F','i','b','r','a','\0'},
    {'P','l','a','s','t','i','c','o','\0'}};
```

Matriz

- O que acontece na memória quando declaramos o arranjo?




```
char nomes[4][10] = {{'A','l','v','e','n','a','r','i','a','\0'},
    {'V','i','n','i','l','\0'}, {'F','i','b','r','a','\0'},
    {'P','l','a','s','t','i','c','o','\0'}};
```

Matriz

- Existe outro modo de inicializar uma matriz

```
char nomes[4][10] =
 {{'A','l','v','e','n','a','r','i','a','\0'},
 {'V','i','n','i','l','\0'},
 {'F','i','b','r','a','\0'},
 {'P','l','a','s','t','i','c','o','\0'}};
```

Matriz

- Existe outro modo de inicializar uma matriz

```
char nomes[4][10] =
 {{'A','l','v','e','n','a','r','i','a','\0'},
 {'V','i','n','i','l','\0'},
 {'F','i','b','r','a','\0'},
 {'P','l','a','s','t','i','c','o','\0'}};

char nomes[4][10];
```

Matriz

- Existe outro modo de inicializar uma matriz
- No segundo modo, cada caractere precisa ser atribuído assim:

```
nomes[0][0] = 'A';
nomes[0][1] = 'l';
nomes[0][2] = 'v';
...
```

```
char nomes[4][10] =
 {{'A','l','v','e','n','a','r','i','a','\0'},
 {'V','i','n','i','l','\0'},
 {'F','i','b','r','a','\0'},
 {'P','l','a','s','t','i','c','o','\0'}};

char nomes[4][10];
```

Comparando as inicializações

```
char nomes[4][10] =
 {{'A','l','v','e','n','a','r','i','a','\0'},
 {'V','i','n','i','l','\0'},
 {'F','i','b','r','a','\0'},
 {'P','l','a','s','t','i','c','o','\0'}};
```

```
char nomes[4][10];
```

Comparando as inicializações

```
char nomes[4][10] =
 {{'A','l','v','e','n','a','r','i','a','\0'},
 {'V','i','n','i','l','\0'},
 {'F','i','b','r','a','\0'},
 {'P','l','a','s','t','i','c','o','\0'}};
```

```
char nomes[4][10];
```

- Usada quando temos poucos dados e os conhecemos de antemão

Comparando as inicializações

```
char nomes[4][10] =
 {{'A','l','v','e','n','a','r','i','a','\0'},
 {'V','i','n','i','l','\0'},
 {'F','i','b','r','a','\0'},
 {'P','l','a','s','t','i','c','o','\0'}};
```

- Usada quando temos poucos dados e os conhecemos de antemão

```
char nomes[4][10];
```

- Usada quando temos muitos dados ou não os conhecemos de antemão

Arranjo de Arranjos

- A outra forma de organizarmos os nomes é por meio de um arranjo de arranjos, alocado dinamicamente

```
int main() {

  char** nomes2 = (char**) malloc(sizeof(char*)*4);

  nomes2[0] = (char*) malloc(sizeof(char)*10);
  nomes2[1] = (char*) malloc(sizeof(char)*6);
  nomes2[2] = (char*) malloc(sizeof(char)*6);
  nomes2[3] = (char*) malloc(sizeof(char)*9);
  ...
```

Arranjo de Arranjos

- A outra forma de organizarmos os nomes é por meio de um arranjo de arranjos, alocado dinamicamente
  - O primeiro arranjo terá quatro elementos (quatro referências a arranjos de caracteres)

```
int main() {

  char** nomes2 = (char**) malloc(sizeof(char*)*4);

  nomes2[0] = (char*) malloc(sizeof(char)*10);
  nomes2[1] = (char*) malloc(sizeof(char)*6);
  nomes2[2] = (char*) malloc(sizeof(char)*6);
  nomes2[3] = (char*) malloc(sizeof(char)*9);
  ...
```

Arranjo de Arranjos

- A outra forma de organizarmos os nomes é por meio de um arranjo de arranjos, alocado dinamicamente
  - O primeiro arranjo terá quatro elementos (quatro referências a arranjos de caracteres)
  - Cada um dos outros arranjos terá um tamanho compatível com a quantidade de caracteres que desejamos armazenar

```
int main() {

  char** nomes2 = (char**) malloc(sizeof(char*)*4);

  nomes2[0] = (char*) malloc(sizeof(char)*10);
  nomes2[1] = (char*) malloc(sizeof(char)*6);
  nomes2[2] = (char*) malloc(sizeof(char)*6);
  nomes2[3] = (char*) malloc(sizeof(char)*9);
  ...
```

Arranjo de Arranjos e Matriz

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  char nomes[4][10] = {{'A','l','v','e','n','a','r','i','a','\0'},
    {'V','i','n','i','l','\0'}, {'F','i','b','r','a','\0'},
    {'P','l','a','s','t','i','c','o','\0'}};

  char** nomes2 = (char**) malloc(sizeof(char*)*4);
  nomes2[0] = (char*) malloc(sizeof(char)*10);
  nomes2[1] = (char*) malloc(sizeof(char)*6);
  nomes2[2] = (char*) malloc(sizeof(char)*6);
  nomes2[3] = (char*) malloc(sizeof(char)*9);

  int x, y;
  for (x=0;x<4;x++) {
    y = -1;
    do {
      y++;
      nomes2[x][y] = nomes[x][y];
    } while (nomes[x][y] != '\0');
  }
  for (x=0;x<4;x++) printf("%s\n",nomes2[x]);
  return 0;
}
```

Arranjo de Arranjos e Matriz

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  char nomes[4][10] = {{'A','l','v','e','n','a','r','i','a','\0'},
    {'V','i','n','i','l','\0'}, {'F','i','b','r','a','\0'},
    {'P','l','a','s','t','i','c','o','\0'}};

  char** nomes2 = (char**) malloc(sizeof(char*)*4);
  nomes2[0] = (char*) malloc(sizeof(char)*10);
  nomes2[1] = (char*) malloc(sizeof(char)*6);
  nomes2[2] = (char*) malloc(sizeof(char)*6);
  nomes2[3] = (char*) malloc(sizeof(char)*9);

  int x, y;
  for (x=0;x<4;x++) {
    y = -1;
    do {
      y++;
      nomes2[x][y] = nomes[x][y];
    } while (nomes[x][y] != '\0');
  }
  for (x=0;x<4;x++) printf("%s\n",nomes2[x]);
  return 0;
}
```

Arranjo de Arranjos e Matriz

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  char nomes[4][10] = {{'A','l','v','e','n','a','r','i','a','\0'},
    {'V','i','n','i','l','\0'}, {'F','i','b','r','a','\0'},
    {'P','l','a','s','t','i','c','o','\0'}};

  char** nomes2 = (char**) malloc(sizeof(char*)*4);
  nomes2[0] = (char*) malloc(sizeof(char)*10);
  nomes2[1] = (char*) malloc(sizeof(char)*6);
  nomes2[2] = (char*) malloc(sizeof(char)*6);
  nomes2[3] = (char*) malloc(sizeof(char)*9);

  int x, y;
  for (x=0;x<4;x++) {
    y = -1;
    do {
      y++;
      nomes2[x][y] = nomes[x][y];
    } while (nomes[x][y] != '\0');
  }
  for (x=0;x<4;x++) printf("%s\n",nomes2[x]);
  return 0;
}
```

Saída

```
Alvenaria
Vinil
Fibra
Plastico
```

Arranjos de Arranjos

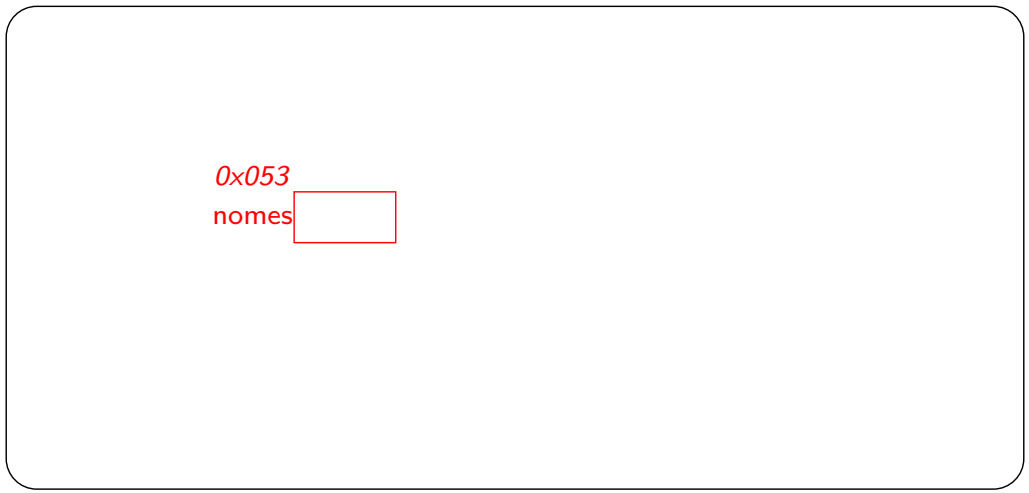
- O que acontece na memória quando declaramos o arranjo?

```
char** nomes = (char**)
                    malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5;
nomes[2] = (char*) malloc(sizeof(char)*5;
nomes[3] = (char*) malloc(sizeof(char)*8;
```

Arranjos de Arranjos

- O que acontece na memória quando declaramos o arranjo?
- Primeiro alocamos espaço para a variável `nomes`

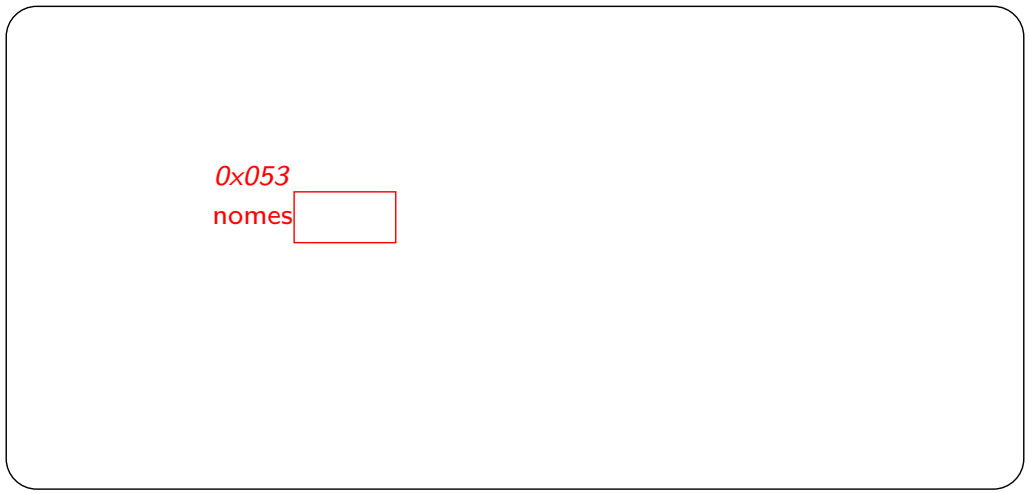
```
char** nomes = (char**)
                    malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5;
nomes[2] = (char*) malloc(sizeof(char)*5;
nomes[3] = (char*) malloc(sizeof(char)*8;
```



Arranjos de Arranjos

- O que acontece na memória quando declaramos o arranjo?
- Primeiro alocamos espaço para a variável `nomes`
  - Conterá o endereço do arranjo correspondente ao arranjo de arranjos

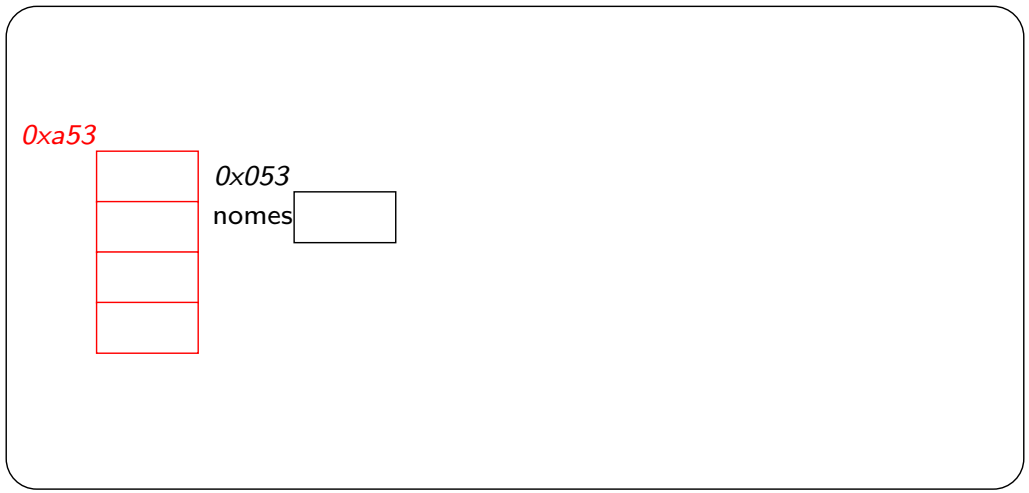
```
char** nomes = (char**)
                    malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5;
nomes[2] = (char*) malloc(sizeof(char)*5;
nomes[3] = (char*) malloc(sizeof(char)*8;
```



Arranjos de Arranjos

- Em seguida alocamos espaço para o arranjo de arranjos

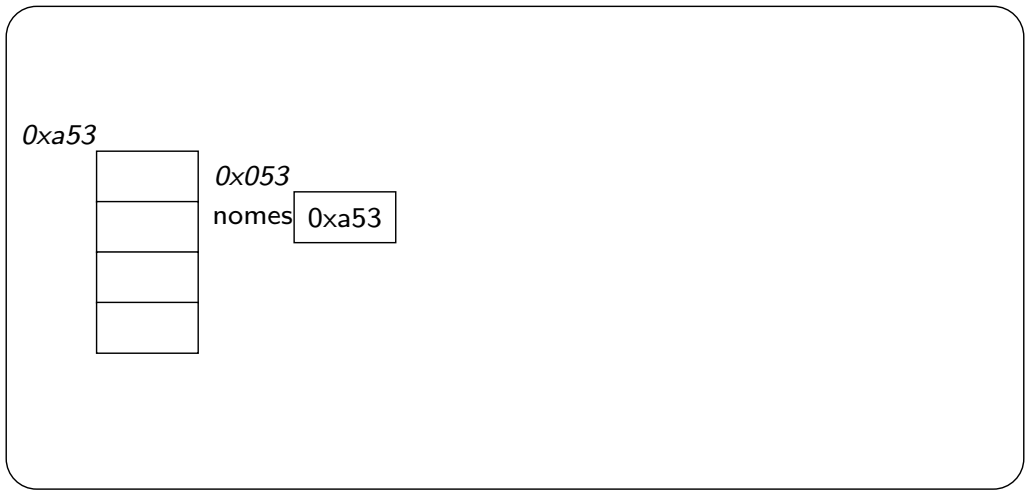
```
char** nomes = (char**)
                        malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```



Arranjos de Arranjos

- Em seguida alocamos espaço para o arranjo de arranjos
  - Cada posição terá espaço suficiente para um endereço (32 ou 64 bits)

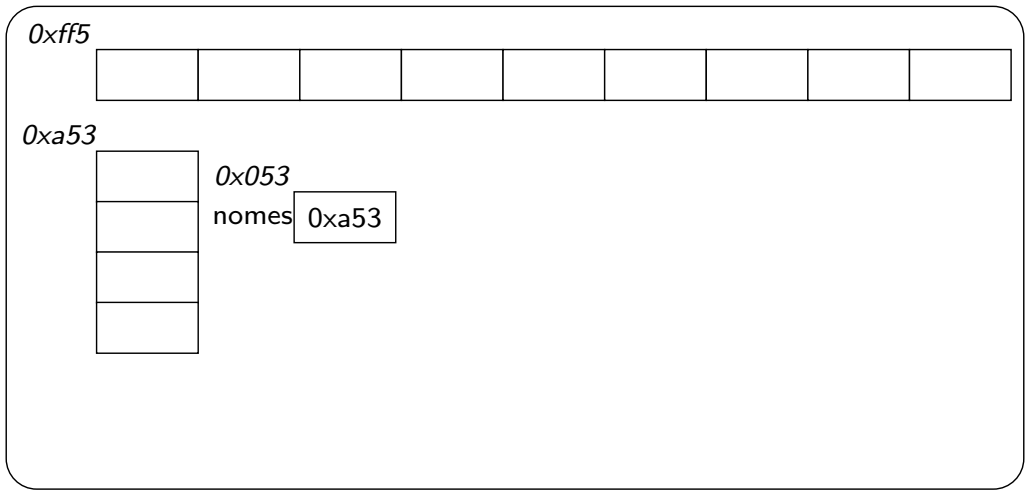
```
char** nomes = (char**)
                        malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```



Arranjos de Arranjos

- Em seguida alocamos espaço para o arranjo de arranjos
  - Cada posição terá espaço suficiente para um endereço (32 ou 64 bits)
- Alocamos então espaço para os arranjos que compõem o arranjo de arranjos

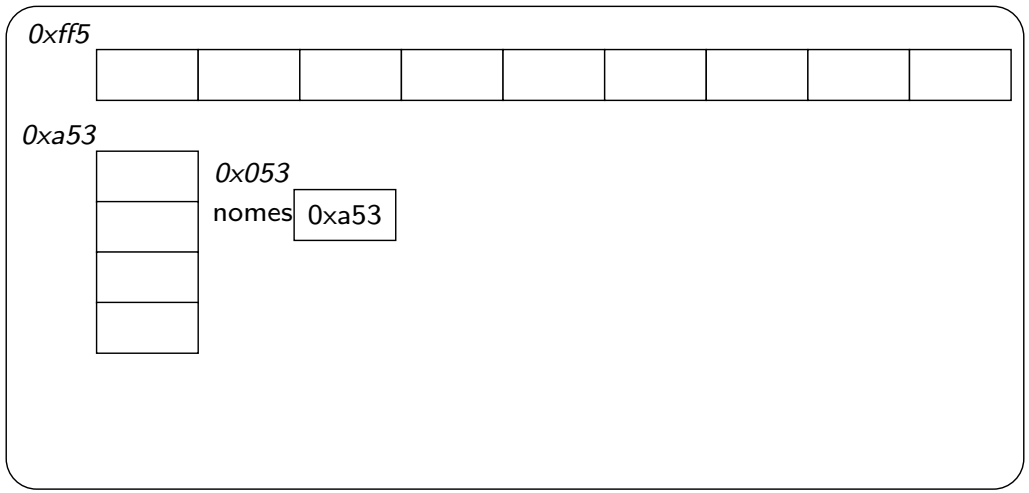
```
char** nomes = (char**)
                        malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```



Arranjos de Arranjos

- Note que não há variável que guarde seus endereços

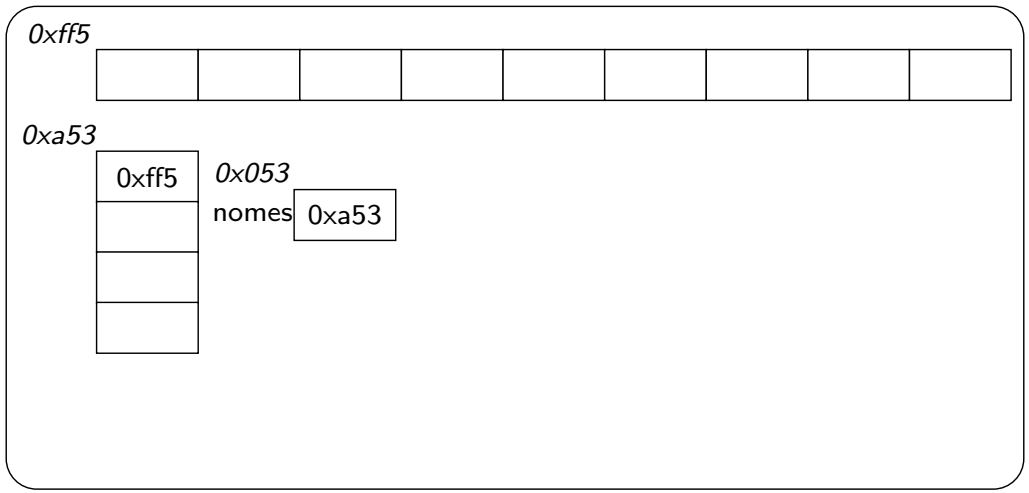
```
char** nomes = (char**)
                    malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```



Arranjos de Arranjos

- Note que não há variável que guarde seus endereços
- Em seguida, guardamos seus endereços nas posições de `nomes`

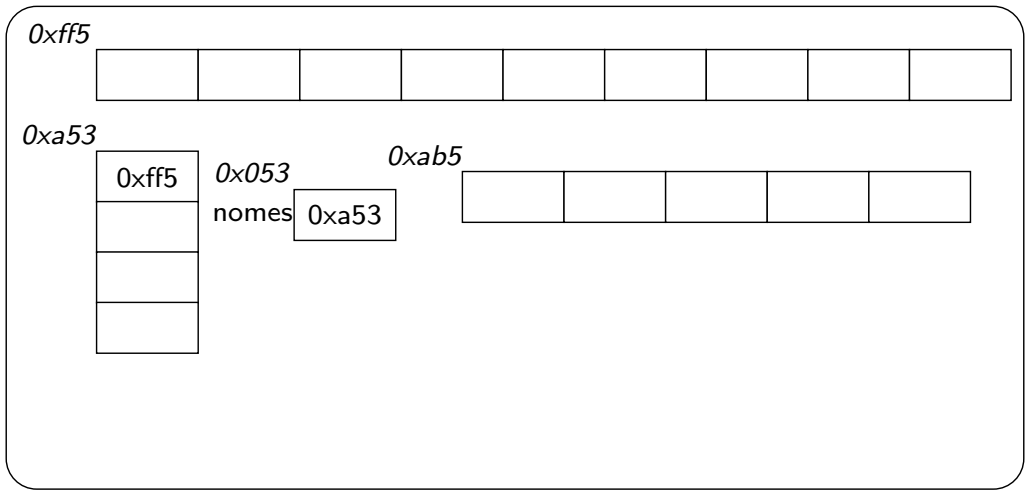
```
char** nomes = (char**)
                    malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```



Arranjos de Arranjos

- Note que não há variável que guarde seus endereços
- Em seguida, guardamos seus endereços nas posições de `nomes`

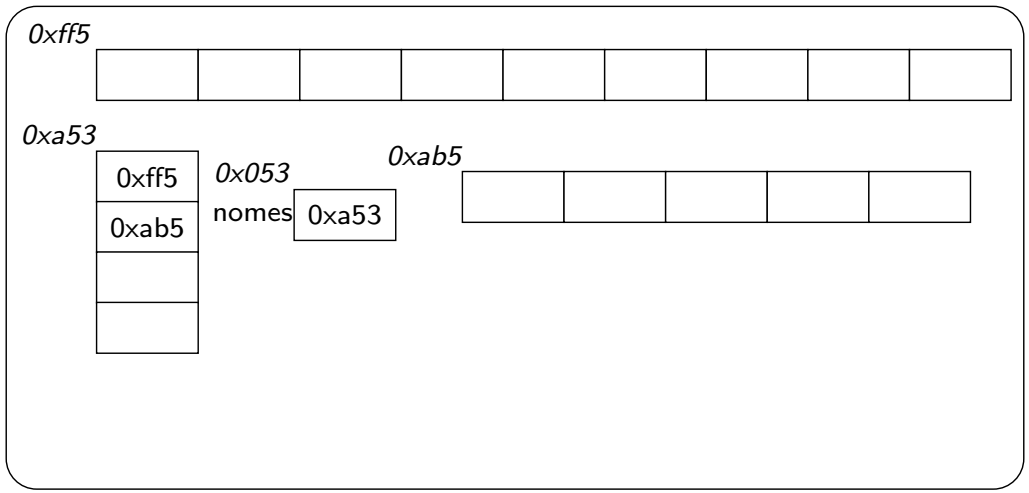
```
char** nomes = (char**)
                    malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```



Arranjos de Arranjos

- Note que não há variável que guarde seus endereços
- Em seguida, guardamos seus endereços nas posições de `nomes`

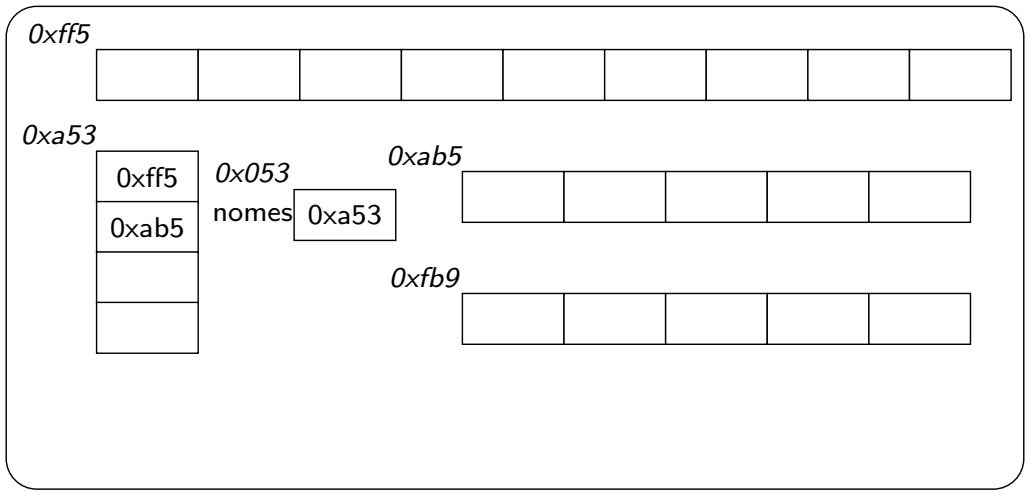
```
char** nomes = (char**)
                    malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```



Arranjos de Arranjos

- Note que não há variável que guarde seus endereços
- Em seguida, guardamos seus endereços nas posições de `nomes`

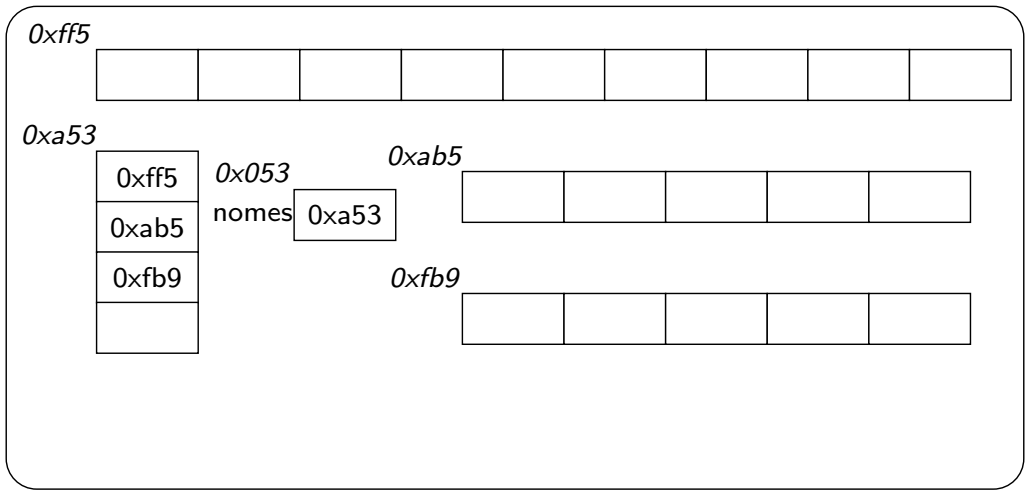
```
char** nomes = (char**)
                    malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```



Arranjos de Arranjos

- Note que não há variável que guarde seus endereços
- Em seguida, guardamos seus endereços nas posições de `nomes`

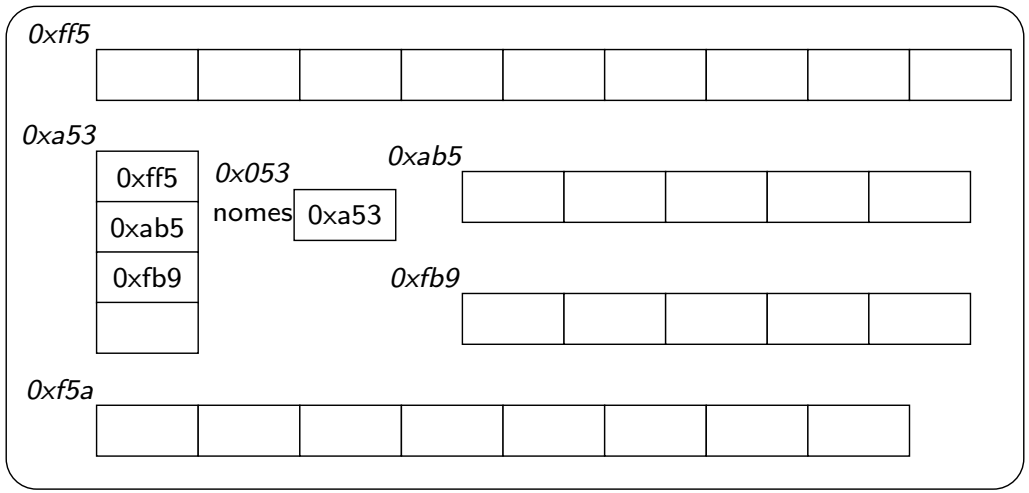
```
char** nomes = (char**)
                    malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```



Arranjos de Arranjos

- Note que não há variável que guarde seus endereços
- Em seguida, guardamos seus endereços nas posições de `nomes`

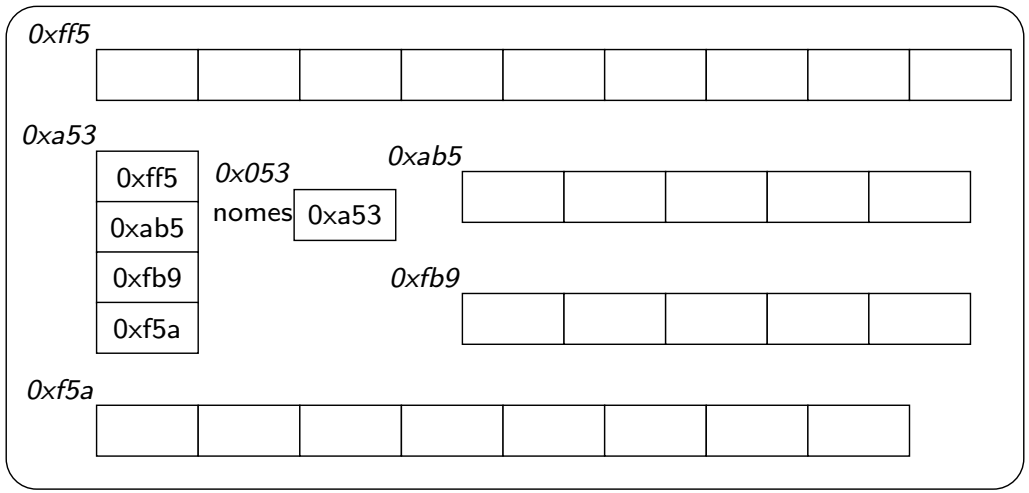
```
char** nomes = (char**)
                    malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```



Arranjos de Arranjos

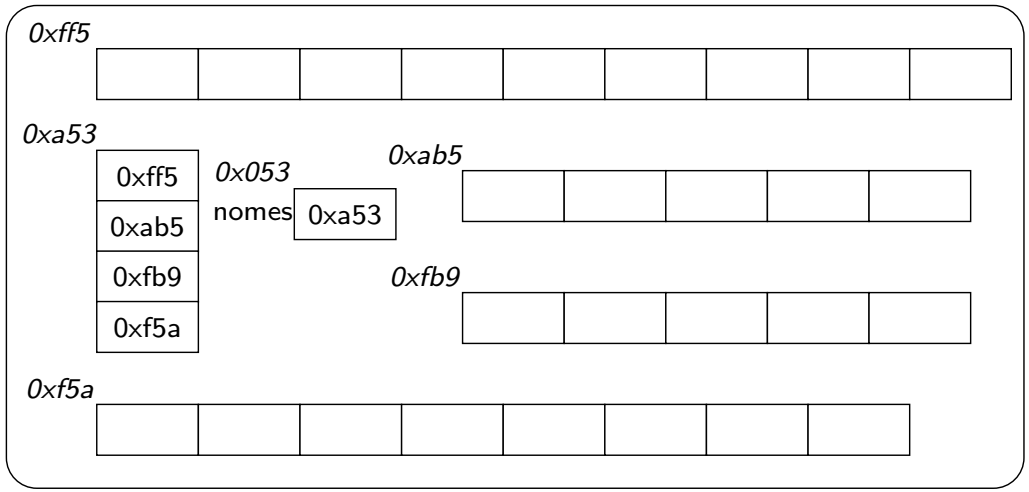
- Note que não há variável que guarde seus endereços
- Em seguida, guardamos seus endereços nas posições de `nomes`

```
char** nomes = (char**)
                    malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```



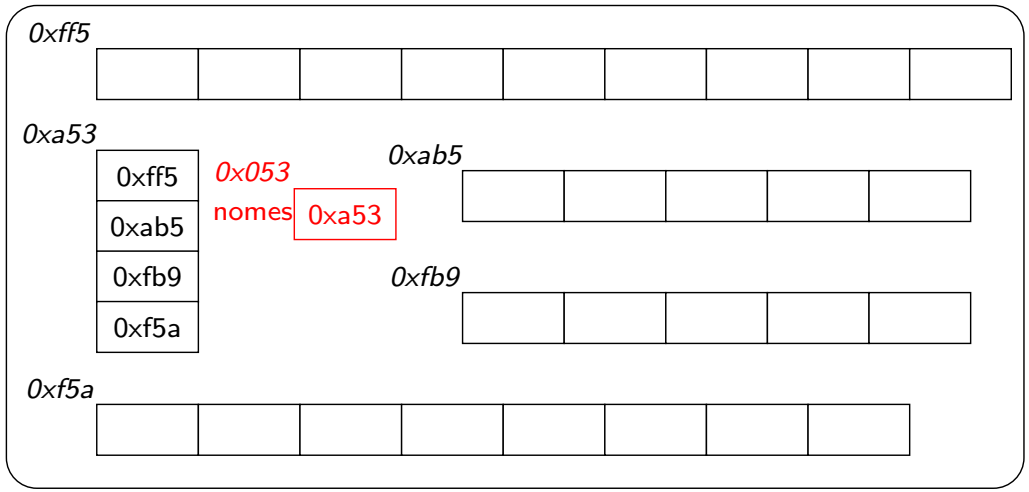
Arranjos de Arranjos

- E o que acontece na memória quando fazemos `nomes[1][2]`?



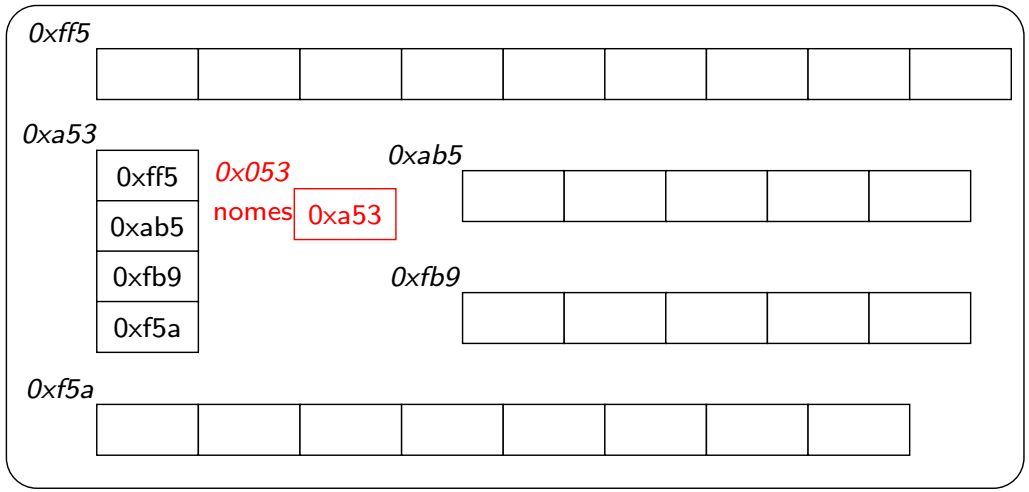
Arranjos de Arranjos

- E o que acontece na memória quando fazemos `nomes[1][2]`?
- O conteúdo da variável `nomes` é lido



Arranjos de Arranjos

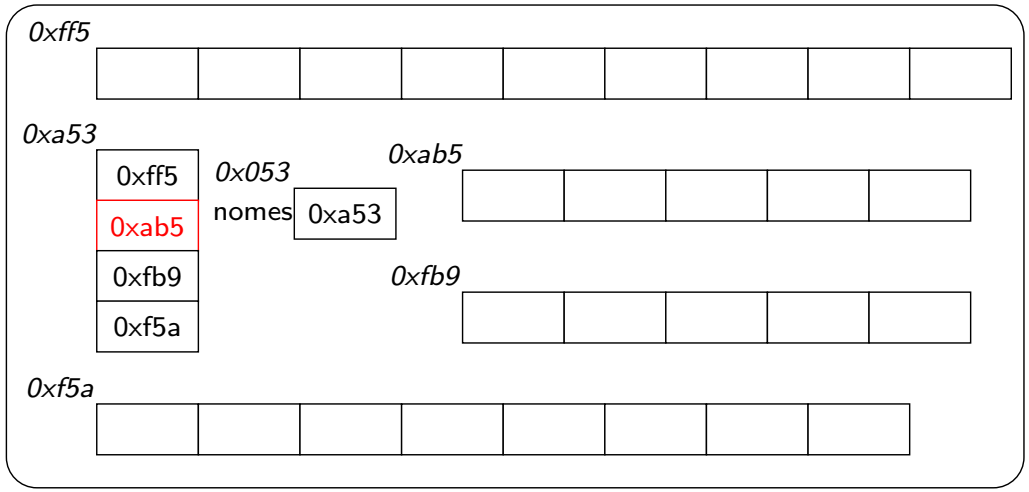
- E o que acontece na memória quando fazemos `nomes[1][2]`?



- O conteúdo da variável `nomes` é lido
- O deslocamento é calculado:
  - $0xa53 + 1 \times 8$, sendo 8 (bytes) o tamanho do endereço

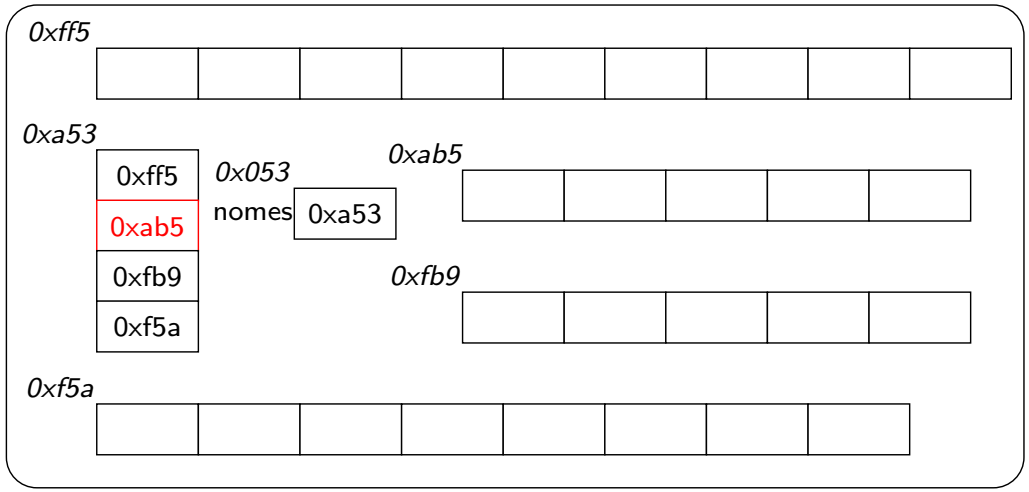
Arranjos de Arranjos

- A região de memória correspondente a esse novo endereço é lida



Arranjos de Arranjos

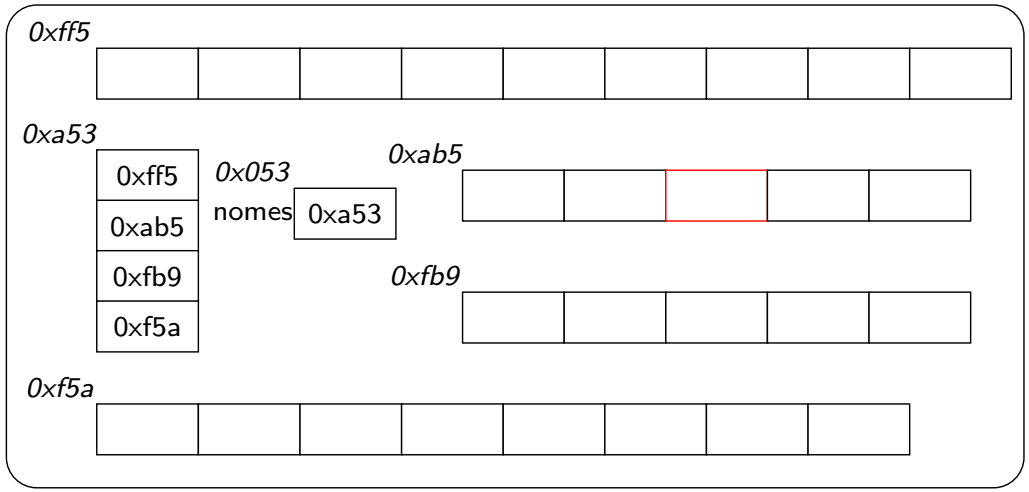
- A região de memória correspondente a esse novo endereço é lida



- O novo deslocamento é calculado
  ($0xab5 + 2 \times 1$, sendo 1 o tamanho do char)

Arranjos de Arranjos

- A região de memória correspondente a esse novo endereço é lida

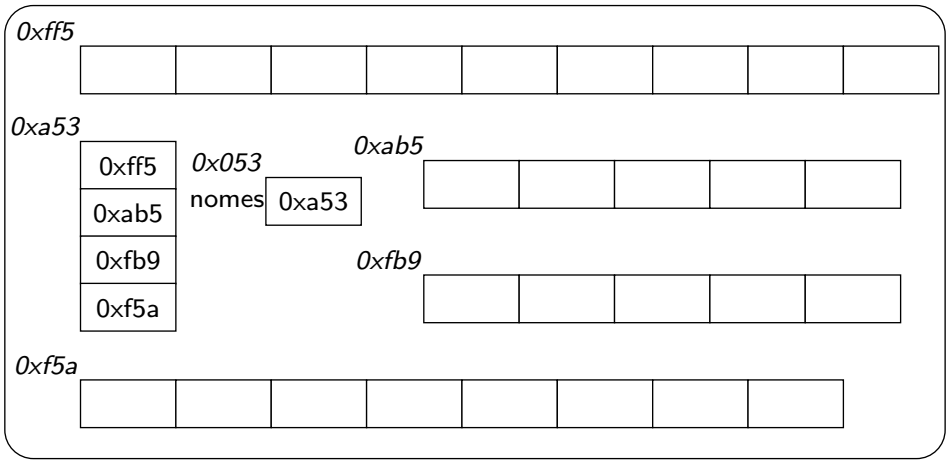


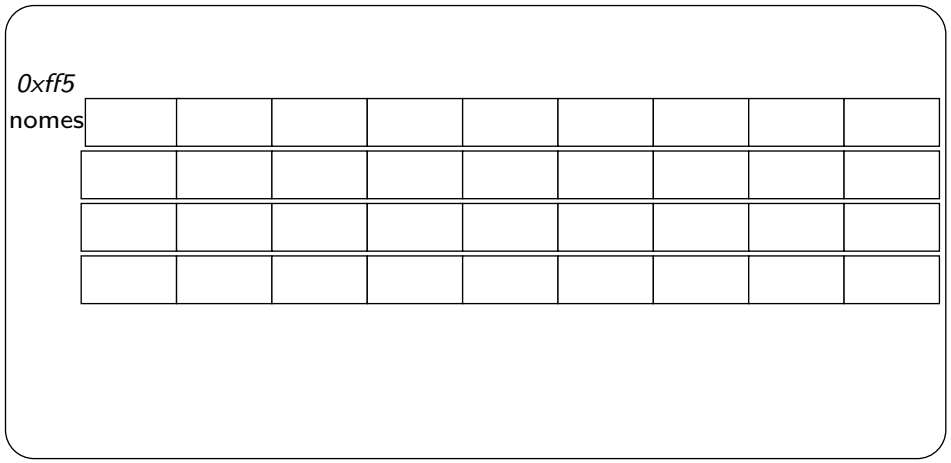
- O novo deslocamento é calculado
  ($0xab5 + 2 \times 1$, sendo 1 o tamanho do char)
- Finalmente, esse novo endereço é visitado, e o seu conteúdo lido

Comparando o uso de memória

```
char** nomes = (char**) malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```

```
char nomes[4][9];
```

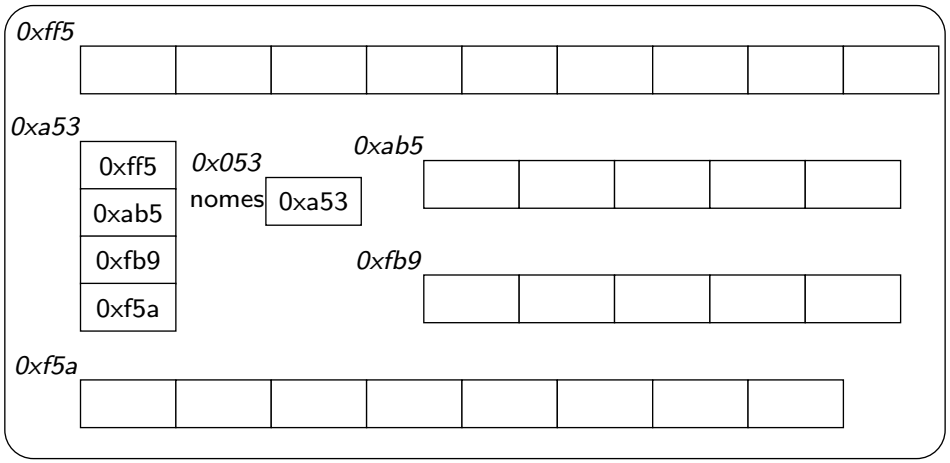


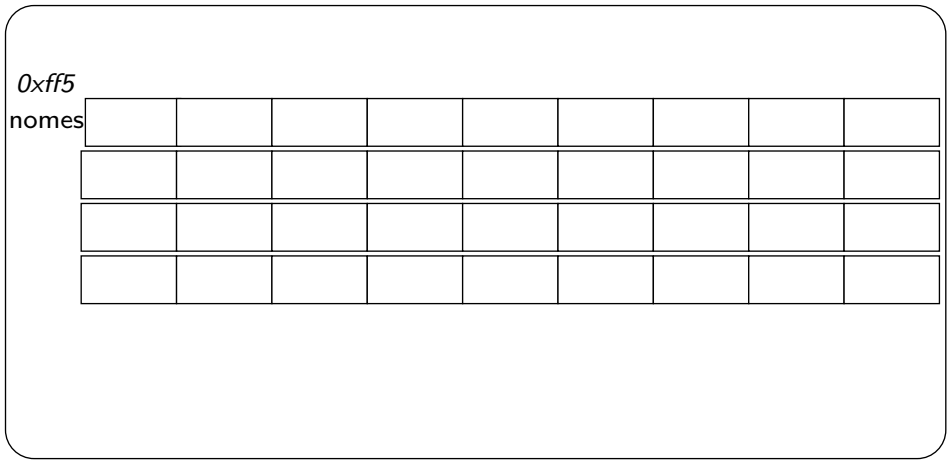


Comparando o uso de memória

```
char** nomes = (char**) malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```

```
char nomes[4][9];
```



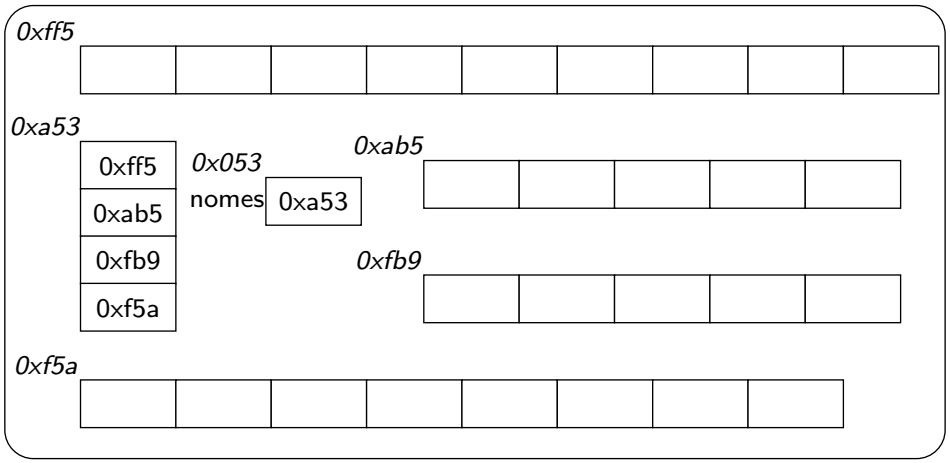


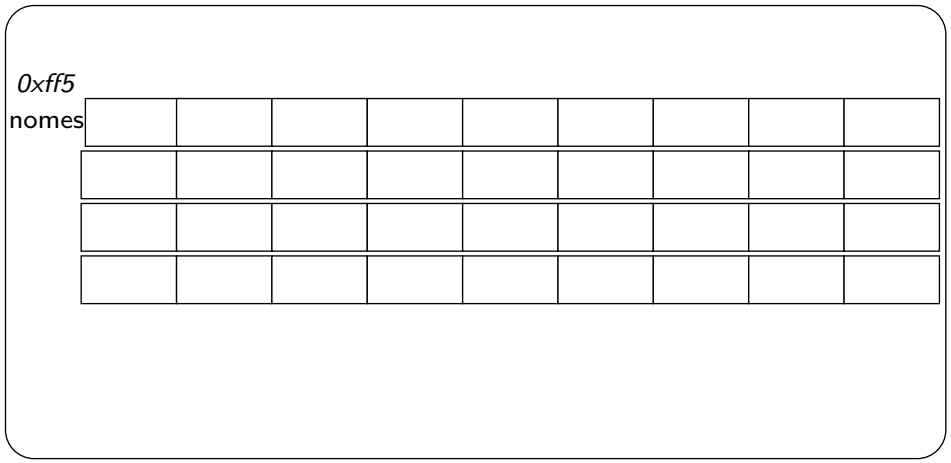
- Memória pode ser alocada de forma dinâmica de acordo com a necessidade

Comparando o uso de memória

```
char** nomes = (char**) malloc(sizeof(char*)*4);
nomes[0] = (char*) malloc(sizeof(char)*9);
nomes[1] = (char*) malloc(sizeof(char)*5);
nomes[2] = (char*) malloc(sizeof(char)*5);
nomes[3] = (char*) malloc(sizeof(char)*8);
```

```
char nomes[4][9];
```





- Memória pode ser alocada de forma dinâmica de acordo com a necessidade
- Memória alocada de uma vez (pode haver desperdício)

Aula 22 – Matrizes (parte 1)

Norton T. Roman & Luciano A. Digiampietri