Aula 05 - Arrays e matrizes

Assuntos

1. Arrays
2. Matrizes

Do semestre passado....

```
l = list()
l1 = []
l2 = [1, 2, 3, 4, 5]
```

Python possui listas em que podemos agrupar valores (variáveis).

Do semestre passado....

```
lista = [1, 2, 3, 4, 5]
for i in range(0, len(lista)):
    print(i)
```

Podemos percorrer uma lista passando elemento a elemento, começando pelo índice zero.

Do semestre passado....

```
lista = [1, 2, 3, 4, 5]
for elemento in lista:
    print(elemento)
```

`for-each` permite percorrer a lista sem saber o indice do elemento.

Em C...

- Não temos a lista como um tipo do C (mas será implementada em CC4652 - Estrutura de Dados)
- Em C temos os arrays, que não são a mesma coisa, mas podemos usar para resolver alguns dos mesmos problemas
- Os arrays começam a contar a partir do zero (como em Python)

```
int main(int argc, char *argv[])
```

Array vs lista

1. Array tem tamanho definido, em listas podemos adicionar e remover valores
2. Array ocupa um espaço contíguo (sequencial, com todos os elementos reservando o mesmo tamanho) na memória. Listas alocam o espaço conforme precisam.

Array vs lista

3. Array em C possuem tipo e só permitem valores do mesmo tipo. Lista em Python podem receber valores de tipos diferentes (mas isto depende da linguagem);

Array em C

```
tipo nome_da_variável [tamanho_do_array];
```

```c
#include <stdio.h>

int main(){
  int array[10];

  for(int i=0; i < 10; i++)
    printf("%d ", array[i]);
}
```

Usando uma variável como tamanho do array

```
#include <stdio.h>

int main(){
  int len = 10
  int array[len];

  for(int i=0; i < len; i++)
    printf("%d ", array[i]);
}
```

Inicialização de arrays

Assim como as variaveis que declaramos até agora, os valores de um array recebem o lixo de memória quando são incializados. Mas podemos inicializar de algumas formas:

1. Declarando cada variável
2. Passando os valores do array
3. Iterando por todas as variáveis

Inicialização: declarando cada variável

```
int array[5];
array[0] = 1;
array[1] = 2;
array[2] = 3;
array[3] = 4;
array[4] = 5;
```

Quais vantagens e desvantagens deste método?

Inicialização: declarando cada variável

```
int array[] = {1, 2, 3, 4, 5};
```

1. Não precisamos do tamanho do array (será usada a quantidade de valores)
2. Os valores devem estas entre chaves (`{` e `}`) para serem usados

Quais vantagens e desvantagens deste método?

Inicialização: iterando pelo array

```
int len = 10;
int array[len];

for(int i = 0; i < len; i++)
    array[i] = 0;
```

Quais vantagens e desvantagens deste método?

Detalhes/cuidados sobre arrays...

1. Se um array ocupa estaço contíguo na memória, então temos que garantir que temos espaço suficiente para armazenar o array;
2. C não possui `for-each` e não temos um método para saber o tamanho do array. Ou seja, quem desenvolve o programa é responsável por saber onde o array termina;
3. Pelo mesmo motivo, podemos ler mais do que inicializado em um array;

Exemplo ruim

```
int len = 10; // comprimento do array
int array[len]; // usado na inicialização

// usado na iteração para inicializar
for(int i = 0; i < len; i++)
    array[i] = 0;

// não usado na iteração para leitura
for(int i = 0; i < 15; i++)
    printf("%d ", array[i]);
```

Exemplo pior ainda

```
int len = 10; // comprimento do array
int array[len]; // usado na inicialização

// não usado na iteração para inicializar
for(int i = 0; i < 15; i++)
    array[i] = 0;

// não usado na iteração para leitura
for(int i = 0; i < 20; i++)
    printf("%d ", array[i]);
```

Por que devemos evitar isso?

Portanto...

Em C temos que controlar a memória usada pelo programa e tomar cuidado:

- Para não escrever em um espaço que não foi alocado para a variável
- Para não ler de um espaço que não foi alocado para a variável

Os dois casos podem dar erros durante a execução ou gerar valores que não podemos confiar que estão corretos.

Exemplo

```
int len;
printf("Digite o tamanho do array:");
scanf("%d", &len);

int array[len];
for(int i = 0; i < len; i++){
    printf("Digite o valor para a posição %d: ", i);
    scanf("%d", &array[i]);
}
printf("Os valores digitados foram:");
for(int i = 0; i < len; i++)
    printf("%d ", array[i]);
```

Matrizes

- São como arrays
- Mas com uma dimensão a mais

```
int matriz[10][10] // matriz de 10x10
```

- Exemplo do uso: imagem em PB pode ser descrita como uma matriz de 2 dimensões. Uma para a quantidade de pixels na horizontal, a outra para a quantidade de pixels na vertical e o valor na posição é a intensidade do pixel

Matrizes

Tudo o que já sabemos sobre vetores se aplica a matrizes:

- Índice começa em zero;
- Valores não são inicializados automaticamente;
- São passados por referência e não por valor;
- A partir do C99, podemos usar variáveis como tamanho de uma matriz (`int v[tamanho]`);
- Alocação de memória contígua;

Matrizes

```
#include <stdio.h>
int main(){
    int linhas = 10;
    int colunas = 10;
    int matriz[linhas][colunas];

    for(int linha=0; linha<linhas; linha++){
        for(int coluna=0; coluna<colunas; coluna++)
            printf("%d\t", matriz[linha][coluna]);
        printf("\n");
    }
}
```

Matrizes

- Podemos pensar em uma matriz como um array de arrays
- O primeiro `for` percorre a matriz em uma direção (por exemplo, em linhas)
- O segundo `for` percorre a matriz na segunda direção (por exemplo, em colunas)

Array de arrays...

- Se uma matriz é um array de array, então podemos ter um array de array de array?

```
int ndarray[][][];
```

Array de arrays

- São muito usados para agrupar dados do mesmo tipo ou relacionados a uma mesma variável
- Por exemplo, uma imagem é tratada como um array de 3 dimensões:
  - Ponto em X
  - Ponto em Y
  - Intensidade da cor em RGB

```
int img[1080][1920][3]; //imagem full HD
```

Array de arrays

- Array de arrays são úteis
- Só temos que tomar cuidado na hora de percorrer esse array, pois pode ser demorado e confuso

```
for(int linha=0; linha<1080; linha++){
    for(int coluna=0; coluna<1920; coluna++){
        for(int cor=0; cor<3; cor++)
            printf("%d ", img[linha][coluna][cor]);
        printf(" ");
    }
    printf("\n");
}
```