4-PONTEIROS EM OUTROS LUGARES

- O que ainda precisamos entender para finalizar ponteiros.

- Vamos criar um progama para ver a quantidade de bytes que uma variavel tem.

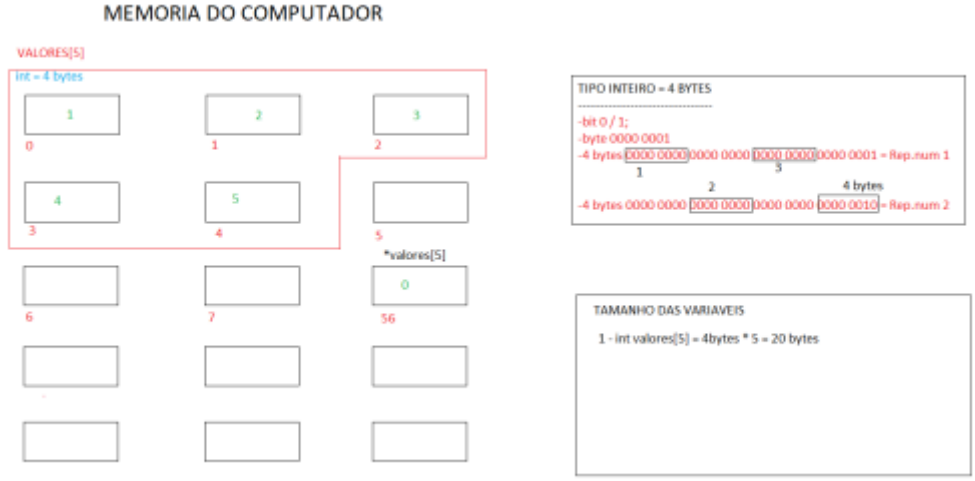
```c
int main(){//inicio_main
    printf("\n");

    int valores[5] = {1, 2, 3, 4, 5}; //declaração_inicialização de um vetor

    for(int i = 0; i < 5; i++){
        printf("O valor %d tem %ld bytes!\n", valores[i], sizeof(valores[i]));
    }

       You, a few seconds ago • Uncommitted changes
    //declarando_variaveis
    //entrada_dados
    //processamento_dados
    //saida_dados
    printf("\n");
    printf("\n");
    return 0;
}//fim_main
```

- Quando estudamos tipo de dados, vimos que cada tipo de dado ocupa um espaço da memoria.



- Vamos percorrer o vetor que ja foi inicializado e imprimir o valor da variavel e seu tamanho em bytes.

sizeof(valores[i]) --- %ld

- Essa função mostra o tamanho da variavel em bytes.

```
O valor 1 tem 4 bytes!
O valor 2 tem 4 bytes!
O valor 3 tem 4 bytes!
O valor 4 tem 4 bytes!
O valor 5 tem 4 bytes!
O valores possui 20 bytes
```

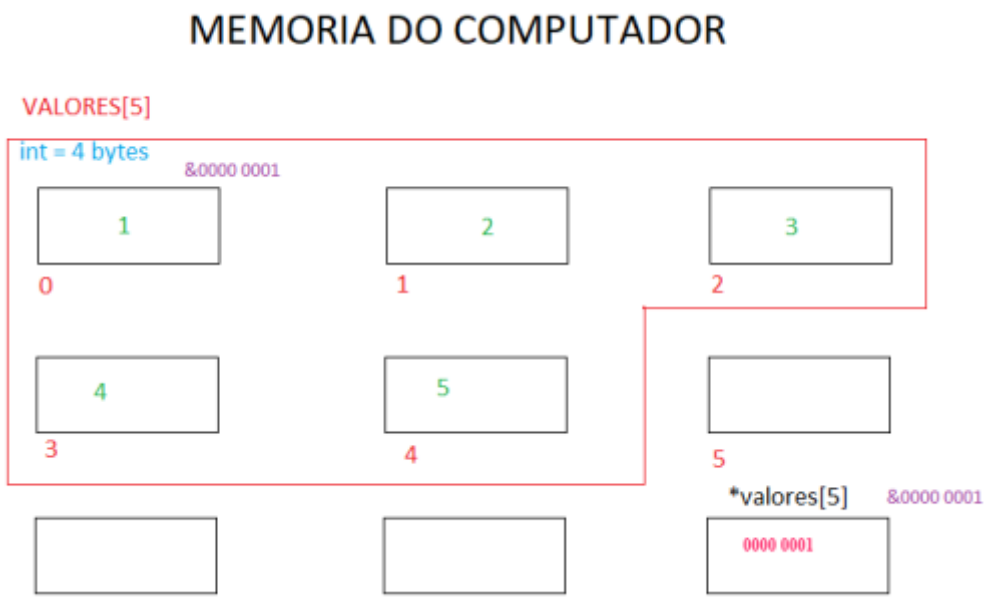
- Vemos que todas possuem 4 bytes pois são do tipo inteiro(figura a cima).

PONTEIROS

- Ao criar um vetor, a linguagem cria um ponteiro com o nome desse vetor e seu endereço. Logo podemos acessar esse vetor pelo ponteiro..

```
//utilização do ponteiro do vetor[5]
printf("valores[0] vale %d e endereço de memoria eh %p\n",valores[0], valores[0]);
printf("*valores vale %d e endereço de memoria eh %p\n",*valores, *valores);
```

```
O valor 1 tem 4 bytes!
O valor 2 tem 4 bytes!
O valor 3 tem 4 bytes!
O valor 4 tem 4 bytes!
O valor 5 tem 4 bytes!
O valores possui 20 bytes
valores[0] vale 1 e endere├ºo de memoria eh 00000001
*valores vale 1 e endere├ºo de memoria eh 00000001
```



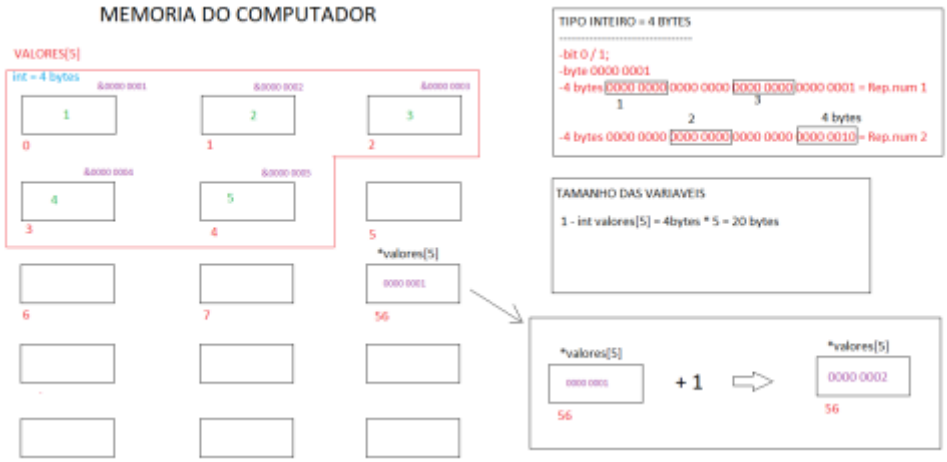
- Para acessar as outras posições

```
// avançando as posições
printf("*(valores + 1) vale %d e endereco de memoria eh %p\n",*(valores+1), *(valores+1));
printf("*(valores + 2) vale %d e endereco de memoria eh %p\n",*(valores+2), *(valores+2));
printf("*(valores + 3) vale %d e endereco de memoria eh %p\n",*(valores+3), *(valores+3));
printf("*(valores + 4) vale %d e endereco de memoria eh %p\n",*(valores+4), *(valores+4));
```

```
*valores vale 1 e endereco de memoria eh 00000001
*(valores + 1) vale 2 e endereco de memoria eh 00000002
*(valores + 2) vale 3 e endereco de memoria eh 00000003
*(valores + 3) vale 4 e endereco de memoria eh 00000004
*(valores + 4) vale 5 e endereco de memoria eh 00000005
```

- Estamos somando ao conteudo do ponteiro(end_vetor) a 1, ou seja, o proximo endereço, percorrendo assim o vetor utilizando o ponteiro.

- Quando fazemos a inicialização de um vetor, o computador aloca um espaço de memoria continuo(um do lado do outro)


MEMORIA DO COMPUTADOR
VALORES[5]
int = 4 bytes
&0000 0001
&0000 0002
&0000 0003
&0000 0004
&0000 0005
1
2
3
4
5
0
1
2
3
4
5
6
7
56
*valores[5]
0000 0001
TIPO INTEIRO = 4 BYTES
-bit 0 / 1;
-byte 0000 0001
-4 bytes 0000 0000 0000 0000 0000 0000 0000 0001 = Rep.num 1
1
2
3
4 bytes
-4 bytes 0000 0000 0000 0000 0000 0000 0000 0010 = Rep.num 2
TAMANHO DAS VARIAVEIS
1 - int valores[5] = 4bytes * 5 = 20 bytes
*valores[5]
0000 0001
56
+1
*valores[5]
0000 0002
56

-