



DEPARTAMENTO
DE COMPUTAÇÃO

CK0215 - T01 - 2019.1

LABORATÓRIO DE PROGRAMAÇÃO

AULA 03

ALOCÇÃO DINÂMICA DE MEMÓRIA

1. ALGUMAS TÉCNICIDADES:

a) PONTEIRO NULO: ARMAZENA O VALOR ZERO. É GARANTIDO NÃO APONTAR PARA QUALQUER VARIÁVEL OU MEMÓRIA ALOCADA PELO PROGRAMA. É TÍPICAMENTE

USADO PARA INDICAR FALTA DE MEMÓRIA OU OUTRA CONDIÇÃO EXCEPCIONAL. É PROIBIDO DEREFERENCIAR

UM PONTEIRO NULO ("NULL POINTER EXCEPTION").

EM C++, A MANEIRA IDEAL DE DENOTAR O VALOR DE UM PONTEIRO NULO É A EXPRESSÃO `nullptr`, CUJO TIPO É PONTEIRO, E NÃO INTEIRO, COMO `0` OU `NULL`.

b) PONTEIRO PARA VOID ("void *"): DIFERENTEMENTE DE UM PONTEIRO "COMUM", UM PONTEIRO PARA VOID NÃO CARREGA O TIPO DAS VARIÁVEIS APONTADAS.

PONTEIROS PARA VOID SÃO USADOS EM MANIPULAÇÃO DE MEMÓRIA DE MAIS BAIXO NÍVEL, ONDE SE DESEJA APENAS ARMAZENAR ENDEREÇOS DE MEMÓRIA. NÃO

(*)
É PERMITIDO FAZER ARITMÉTICA DE PONTEIROS COM PONTEIROS PARA VOID.

2. DURAÇÃO DE ARMAZENAMENTO: AS SEGUINTE EXISTEM PARA "OBJETOS" NA MEMÓRIA EM C++:

a) ESTÁTICA: DURA DURANTE TODA A EXECUÇÃO DO PROGRAMA. → "VARIÁVEIS GLOBAIS"

b) THREAD: DURA DURANTE TODA A EXECUÇÃO DA THREAD.

→ "VARIÁVEIS LOCAIS"
c) AUTOMÁTICA: DURA ATÉ O FIM DA EXECUÇÃO DO BLOCO CORRESPONDENTE.

d) DINÂMICA: DA ALOCAÇÃO À DESALOCAÇÃO.

3. ALOCAÇÃO DINÂMICA EM C++: É FEITA BASICAMENTE POR MEIO DOS OPERADORES **NEW** E **DELETE**:

a) EXEMPLO PARA UM OBJETO:

```
[ INT *P = NEW INT;  
  ...  
  DELETE P;
```


b) EXEMPLO PARA UM VETOR:

```
INT n; CIN >> n;  
DOUBLE *p = NEW DOUBLE[n];  
...  
DELETE[] p;
```

c) FALTA DE MEMÓRIA: SE NÃO HOUVER MEMÓRIA DISPONÍVEL, O OPERADOR NEW Joga uma exceção de tipo **BAD_ALLOC** (EXATAMENTE OU DELA DERIVADA), QUE DEVE SER TRATADA COM UM TRY-CATCH.

OUTRA MANEIRA É USAR A VERSÃO DO OPERADOR NEW QUE RETORNA UM PONTEIRO NULO:

```
#include <NEW>  
...  
DOUBLE *p = NEW(STD::nothrow)  
                DOUBLE[10];  
...  
DELETE[] p;
```

4. USOS DA ALOCAÇÃO DINÂMICA:

a) QUANDO A DURAÇÃO DESEJADA TRANSCENDE BLOCOS, MAS TAMBÉM NÃO É A DO PROGRAMA TODO (OU THREAD).

b) QUANDO O TAMANHO DESEJADO É MUITO GRANDE E INADEQUADO PARA UMA VARIÁVEL "LOCAL".

6. EXERCÍCIO PARA SALA: ESCREVA UM PROGRAMA QUE REPETIDAMENTE LEIA INTEIROS DO USUÁRIOS, GUARDANDO-OS NUM VETOR ALOCADO DINAMICAMENTE. SEMPRE QUE O VETOR FICAR CHEIO, ALOQUE UM NOVO COM O DOBRO DO TAMANHO (E DESALOQUE O ANTERIOR, CLARO). QUANDO O USUÁRIO DIGITAR

O PRIMEIRO NEGATIVO, TODOS OS NÚMEROS NÃO-NEGATIVOS DIGITADOS DEVEM SER IMPRESSOS, E O PROGRAMA ACABA.

5. EXEMPLO:

```
#include <iostream>
using std::cin; using std::cout;
#include <new>
using std::nothrow;
```



```
DOUBLE *LER_VETOR (INT TAM)
{
    DOUBLE *v = NEW(NOTHROW) DOUBLE[TAM];
    if (v != NULLPTR)
    {
        FOR(INT i=0; i != TAM; ++i)
        {
            COUT << "DIGITE v[" << i << "]: ";
            CIN >> v[i];
        }
    }
    RETURN v;
}
```

```
DOUBLE SOMA (DOUBLE *v, INT TAM)
{
    DOUBLE *fim = v + TAM;
    DOUBLE SOMA = 0;
    FOR(DOUBLE *p=v; p != fim; ++p)
        SOMA += *p;
    RETURN SOMA;
}
```

if (v)

b) QUANDO O TAMANHO DESEJADO É MUITO GRANDE E INADEQUADO PARA UMA VARIÁVEL "LOCAL".

6. EXERCÍCIO PARA SALA: ESCREVA UM PROGRAMA QUE REPETIDAMENTE LEIA INTEIROS DO USUÁRIOS, GUARDANDO-OS NUM VETOR ALOCADO DINAMICAMENTE. SEMPRE QUE O VETOR FICAR CHEIO, ALOQUE UM NOVO COM O DOBRO DO TAMANHO (E DESALOQUE O ANTERIOR, CLARO). QUANDO O USUÁRIO DIGITAR

O PRIMEIRO NEGATIVO, TODOS OS NÚMEROS NÃO-NEGATIVOS DIGITADOS DEVEM SER IMPRESSOS, E O PROGRAMA ACABA. COMECE COM UM VETOR DE TAMANHO 1.

(CONT. DE 5)

```
INT MAIN ()
{
    INT n; DO { COUT << "TAM: "; CIN >> n; } WHILE (n < 1);

    DOUBLE *v = LER_VETOR(n);
    IF (v == NULLPTR) { COUT << "SEM MEMÓRIA!\n";
                      RETURN 1; }

    DOUBLE SOMA = SOMA(v, n);
    COUT << "A SOMA É " << SOMA << '\n';
    DELETE [] v;
}
```