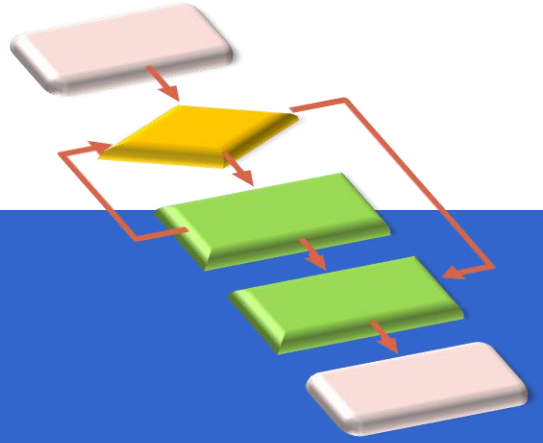
INSTITUTO DE COMPUTAÇÃO



Algoritmos II

ALOCAÇÃO DINÂMICA DE MEMÓRIA
Ponteiros em C

Prof.ª Vanessa de Oliveira Campos

Ponteiros em C

- operador * (conteúdo)
  - Utilizado para:
    - declaração de variáveis do tipo ponteiro de memória.
    - acessar o conteúdo de um endereço apontado por uma variável do tipo ponteiro.
- operador & (endereço)
  - Operador pode ser utilizado para obter o endereço de memória de uma variável.

Ponteiros em C

- Declaração de um ponteiro:

  < tipo apontado > * < nome variável ponteiro >

  onde:

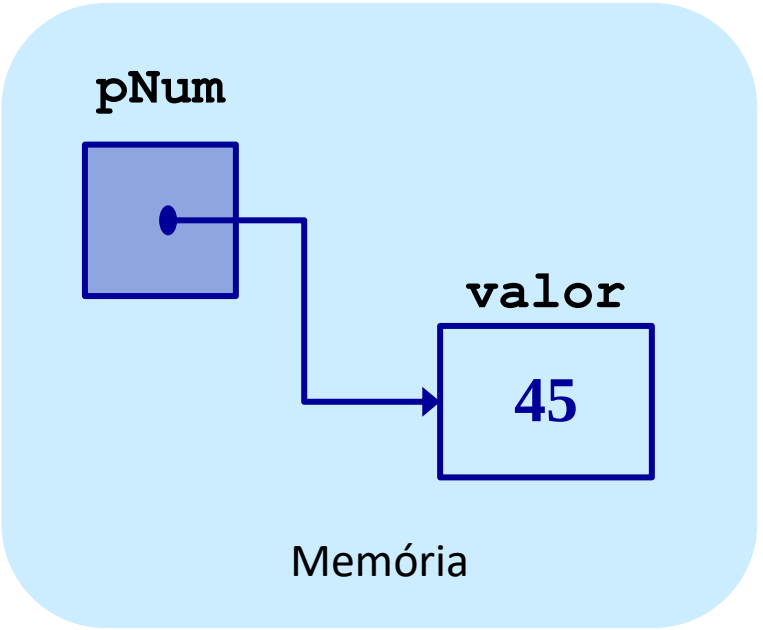
  < nome variável ponteiro > é escolhido tal qual o nome de uma variável simples;

  < tipo apontado > corresponde a um dos tipos válidos na linguagem.

```c
// declaração de uma variável do tipo TAluno
typedef struct TAluno{
    char Nome[30];
    int RGA;
    char Curso[50];
} TAluno;

void main()
{
    int *pNum;
    char *pNome;
    TAluno *pEst ;
    int valor;
    valor = 45;
    pNum = &valor;
    printf("%d", *pNum );
    ...
}
```

Memória		
Nome	End.	Conteúdo
pNum	10	13
pNome	11	5000
pEst	12	343
valor	13	45
...	...	...



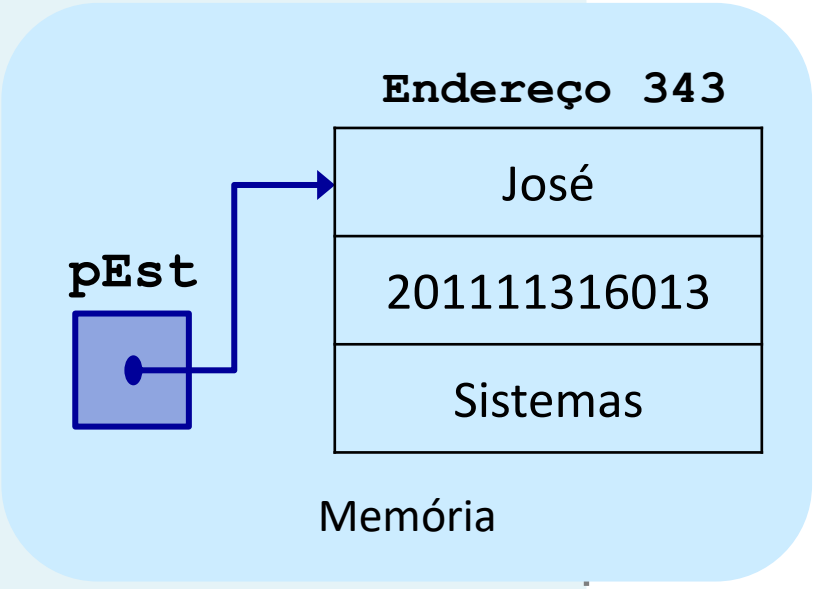
```c
// declaração de uma variável do tipo TAluno
typedef struct TAluno{
    char Nome[30];
    int RGA;
    char Curso[50];
} TAluno;

void main()
{
    int *pNum;
    char *pNome;
    TAluno *pEst ;
    int valor;
    valor = 45;
    pNum = &valor;
    ...
    printf("%d\n", *pNum );
    printf("%s\n", pNome);
    printf("%s - %d - %s \n", pEst->Nome, pEst->RGA, pEst->Curso);
}
```

```c
// declaração de uma variável do tipo TAluno
typedef struct TAluno{
    char Nome[30];
    int RGA;
    char Curso[50];
} TAluno;

void main()
{
    int *pNum;
    char *pNome;
    TAluno *pEst ;
    int valor;
    valor = 45;
    pNum = &valor;
    ...
    printf("%d\n", *pNum );
    printf("%s\n", pNome);
    printf("%s - %d - %s \n", pEst->Nome, pEst->RGA, pEst->Curso);
}
```

Memória		
Nome	End.	Conteúdo
pNum	10	13
pNome	11	5000
pEst	12	343
valor	13	45
...	...	...



Alocação dinâmica de memória

- Com um ponteiro, é possível:
  - Apontar para uma região de memória alocada (dinamicamente ou estaticamente).
  - Alocar uma região de memória dinamicamente.

Alocação dinâmica de memória

- A função **malloc** solicita a alocação de um espaço contínuo de memória.

Sintaxe:

<var do tipo ponteiro> = (tipo *) malloc (tamanho em bytes)

- biblioteca **stdlib**

Exemplo:

```
#include <stdlib.h>
...
   int *p;
   p = (int *)malloc(sizeof(int));
...
```

Alocação dinâmica de memória

- Para verificar se a alocação obteve sucesso: **NULL**
  - Ponteiro utilizado na alocação de memória com valor igual a **NULL** representa uma falha de alocação.

Alocação dinâmica de memória

Exemplo:

```
#include <stdlib.h>
...
   int *p;
   p = (int *)malloc(sizeof(int));
   if (p == NULL)
      printf("Problemas!");
   else
      printf("Ok!");
...
```

Alocação dinâmica de memória

- Uma vez que não sejam mais necessários no algoritmo, os espaços alocados podem ser liberados por meio do comando da função **free**.

Sintaxe:

```
free ( <var do tipo ponteiro> )
```

Alocação dinâmica de memória

Exemplo:

```
#include <stdlib.h>
...
   int *p;
   p = (int *)malloc(sizeof(int));
   if (p == NULL)
      printf("Problemas!");
   else
   {
      // leitura e processamento
      free(p);
   }
...
```

```c
typedef struct TAluno{
    char Nome[30];
    int RGA;
    char Curso[50];
} TAluno; // declaração de uma variável do tipo TAluno

void main()
{
    int *pNum;
    TAluno *pEst;
    ...
    // alocando espaço para usar um vetor com 20 posições
    pNum = (int *) malloc(sizeof(int)* 20);
    if (pNum != NULL)
        ...
    // alocando espaço para um registro
    pEst = (TAluno *) malloc (sizeof(TAluno));
    if ( pEst != NULL )
        ...
    free(pNum); // liberação de memória
    free(pEst); // liberação de memória
}
```

Acesso às variáveis alocadas através ponteiros

- O acesso a uma variável alocada por um ponteiro é feita das seguintes maneiras:

  A) Quando aloca-se um conjunto de posições de memória, está se criando um vetor de um determinado tipo. Neste caso o acesso é semelhante ao uso de variáveis estáticas.

  B) Quando apenas uma posição é alocada utiliza-se o operador * antes do nome da variável para indicar que se está acessando o conteúdo. Também pode-se abrir mão do * caso se esteja lidando com um registro alocado dinamicamente, utilizando-se o operador ->

```c
typedef struct TAluno{
    char Nome[30];
    int RGA;
    char Curso[50];
} TAluno; // declaração de uma variável do tipo TAluno

void main()
{
    int *pNum, n, i;
    TAluno *pEst ;
    n = 100;

    pNum = (int *) malloc(sizeof(int)* n);
    if (pNum != NULL)
        for (i = 0 ; i < n ; i++ )
            scanf("%d", &pNum[i]);
    else
        printf("Não foi possível alocar a memória");

    pEst = (TAluno *) malloc (sizeof(TAluno) * 50);
    if ( pEst != NULL )
        pEst[2].RGA = 2020199187064;
    else
        printf("Não foi possível alocar a memória");
    ...
}
```

```c
typedef struct TAluno{
    char Nome[30];
    int RGA;
    char Curso[50];
} TAluno; // declaração de uma variável do tipo TAluno

void main()
{
    int *pNum, n, i;
    TAluno *pEst ;

    pEst = (TAluno *) malloc (sizeof(TAluno));
    if ( pEst != NULL )
    {
        (*pEst).RGA = 2020199187064;
       pEst->RGA = 2020199187064;
    }
    else
        printf("Não foi possível alocar a memória");
    ...
}
```

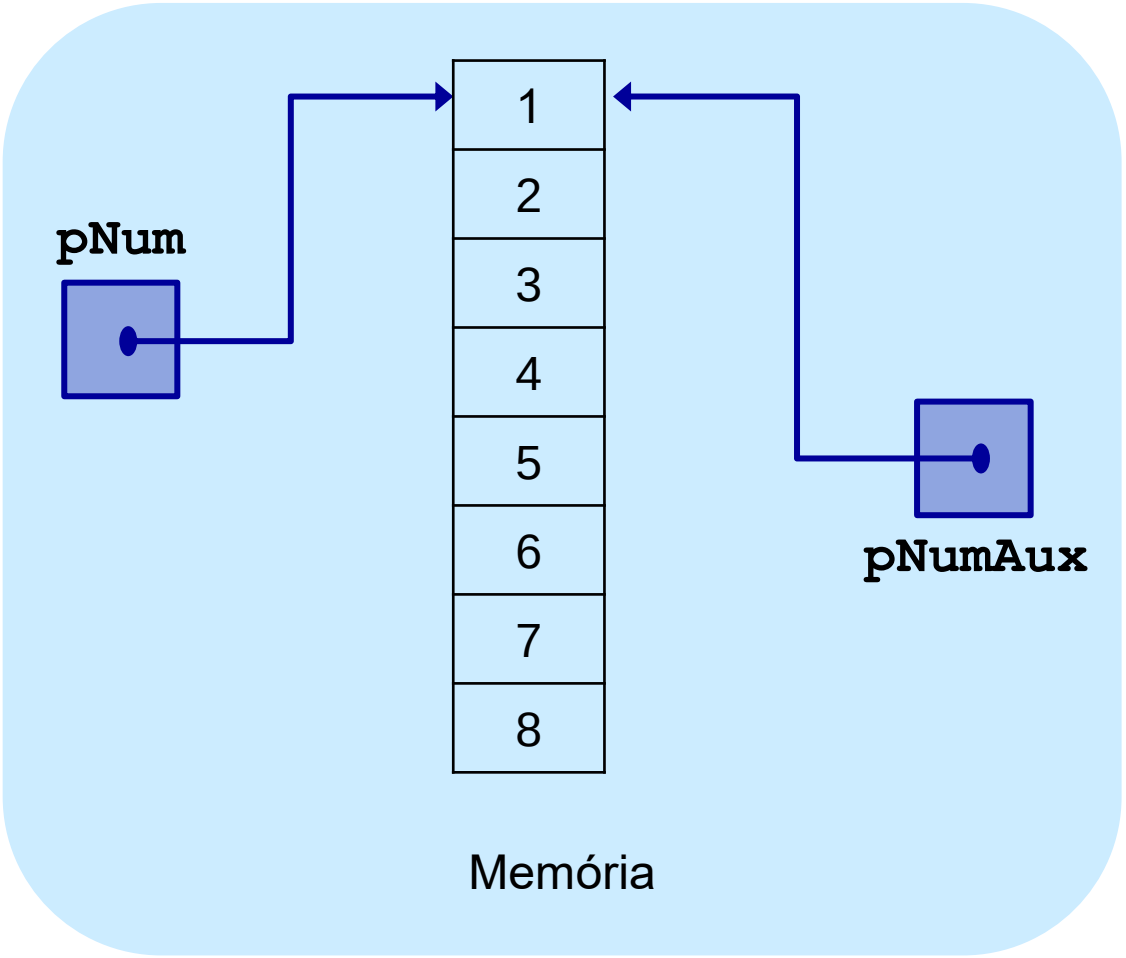
Atribuições

- Sem o uso do operador * antes do nome da variável ponteiro, a alteração é feita sobre o valor do ponteiro (endereço de memória). Portanto:
  - deve-se atribuir endereços; ou
  - pode-se atribuir **NULL**, representando que esse ponteiro não está apontando para lugar algum da memória.
  - pode-se fazer atribuição entre ponteiros, desde que ambos apontem para o mesmo tipo.
    - Neste caso, os dois apontarão para o mesmo endereço físico de memória.

```c
int main()
{
    int *pNum, *pNumAux, i;
    pNumAux = NULL;
    pNum = (int * ) malloc(sizeof(int) * 8 );
    if ( pNum != NULL )
    {
        pNumAux = pNum ;
        for ( i = 0 ; i < 4; i++ )
        {
            pNum[i] = i + 1;
            pNumAux[4 + i]  = i + 5;
        }
        for ( i = 0 ; i < 4; i++ )
            printf("%d - %d \n", pNum[i], pNumAux[4 + i]);
        // como pNum e pNumAux apontam para o mesmo endereço pode-se
        // utilizar qualquer um dos dois para liberar a memória
        free(pNumAux);
    }
}
```

Usando ponteiros para acessar um vetor

Memória		
Nome	End.	Conteúdo
pNum	10	100
pNumAux	11	100
...	...	...
	100	1
	101	2
	102	3
	103	4
	104	5
	105	6
	106	7
	107	8
...	...	...



Exercício de Fixação

1) Quais serão os valores de x, y e p ao final do trecho de código abaixo?

```
int x, y, *p;y = 0;
p = &y;
x = *p;
x = 4;
(*p)++;
--x;
(*p) += x;
```

Exercício de Fixação

2) Faça um programa que:

a) declare uma variável "a" do tipo float;

b) declare um ponteiro "p" para o tipo float;

c) Peça que o usuário digite um número do tipo real, e o armazene em a;

d) Imprima o conteúdo de a e o endereço de a;

e) Imprima o conteúdo do endereço apontado por p e o conteúdo de p;

f) Atribua o endereço de a ao ponteiro p;

g) Imprima o conteúdo do endereço apontado por p e o conteúdo de p.

Exercício de Fixação

3) Faça um algoritmo que leia as notas de um aluno e calcule a média das notas inseridas. Antes de inserir as notas, você deve perguntar ao usuário quantas notas ele deseja inserir para o aluno atual. Depois disso, exiba as notas e a média do aluno.