

Laboratório de Programação I (MATA57)



Prof.: Claudio Junior N. da Silva (claudiojns@ufba.br)

Alocação Dinâmica de Memória

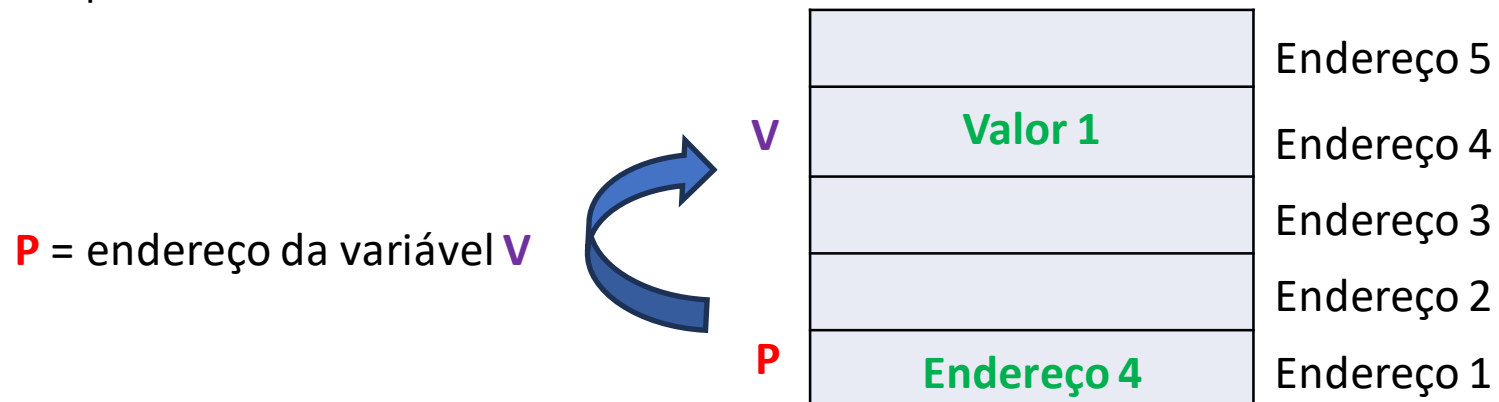
2023.1



Na aula anterior....

Ponteiros

- Toda variável tem um **endereço** ou uma posição associados na **memória**;
- O endereço é visto como um **ponteiro (ou apontador)**. É uma referência para a posição de memória de uma variável;
- Ponteiros fornecem um modo de acesso à variável sem referenciá-la diretamente;
- Um endereço pode ser armazenado em uma variável do tipo ponteiro (ponteiro variável):
 - Ponteiro variável é uma variável que contém o endereço de outra variável:
 - **P** aponta para **V**:



Ponteiros

```
int *p;           // declara p tipo ponteiro para inteiro
int v;           // declara v tipo inteiro
p = &v;         // atribui o endereço de v ao ponteiro p
                  // ou seja, aponta-se p para v
```

```
int i, j;
int *ip;
i = 12;
ip = &i;
j = *ip;
*ip = 21;
```

A variável **ip** armazena um ponteiro para um inteiro

O endereço de **i** é armazenado em **ip**

O conteúdo da posição apontada por **ip** é armazenado em **j**

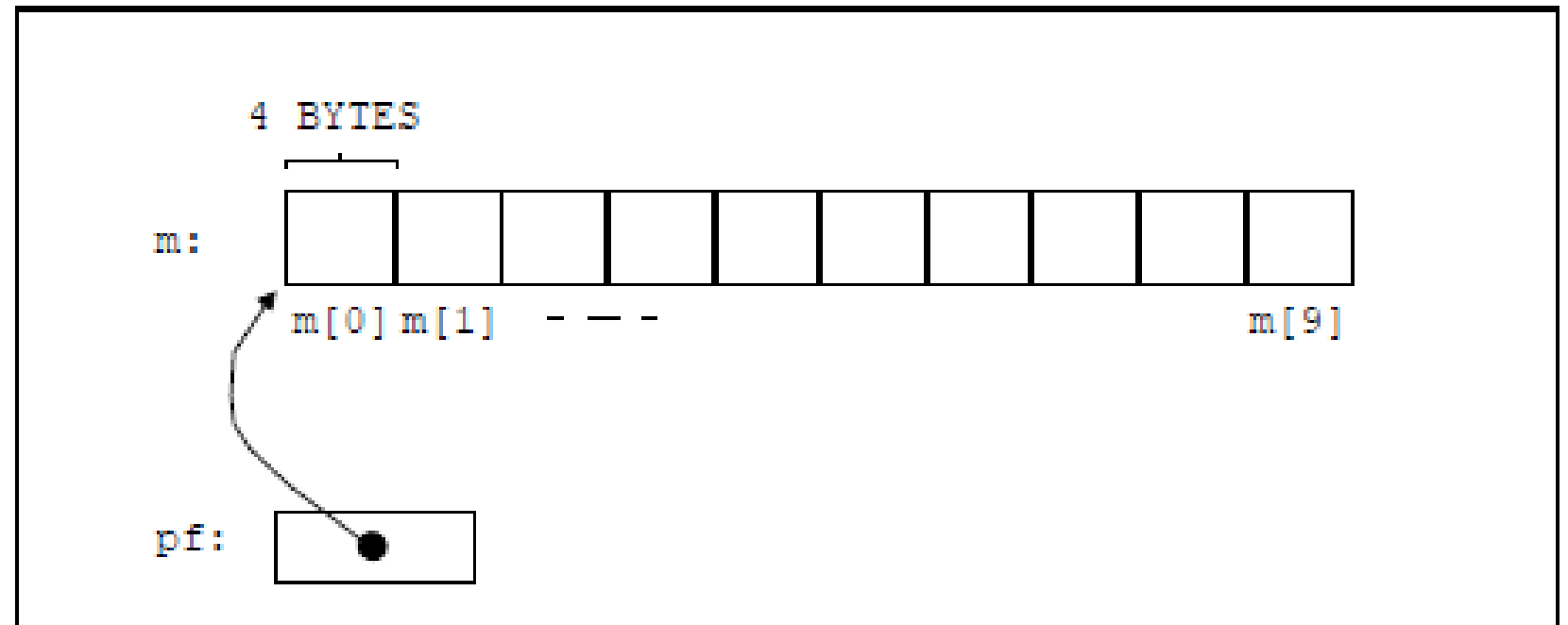
O conteúdo da posição apontada por **ip** passa a ser **21**

Arrays

- Ex:

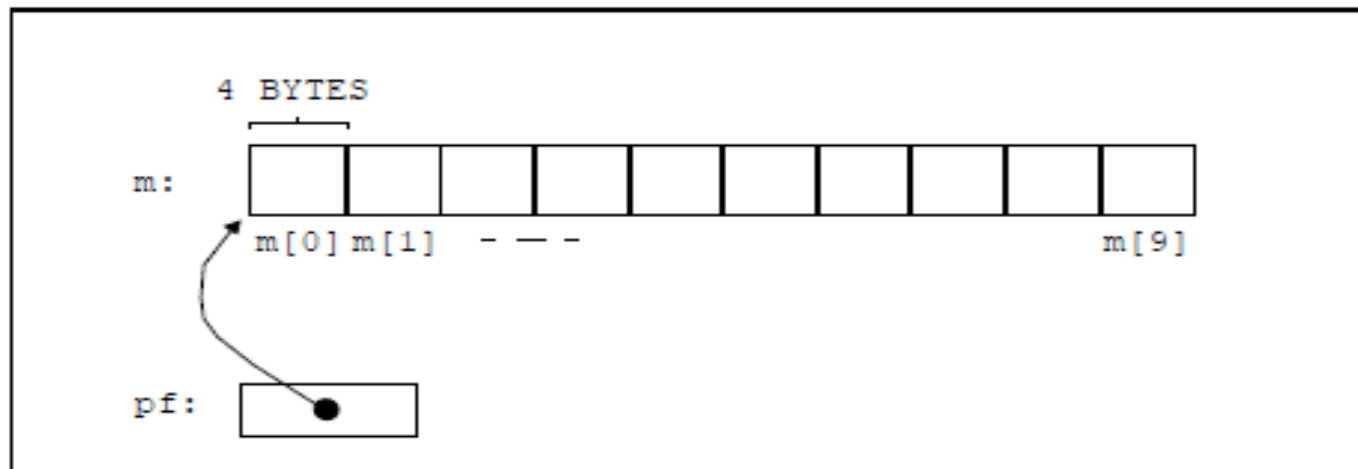
```
float m[10], *pf;
```

```
pf = m;
```



Referenciando Arrays

- `float m[10], n[10]; float *pf;`
- Em `float m[10]`, `m` é uma constante que endereça o primeiro elemento do array
- Portanto, não é possível mudar o valor de `m`



```
m = n;      /* erro: m é constante ! */  
pf = m;     /* ok */
```

Referenciando Elementos de Arrays

- Pode-se referenciar os elementos de um array através de ponteiros:

```
float m[] = { 1.0, 3.0, 5.75, 2.345 };  
float *pf;  pf = &m[2];
```

```
cout << *pf; /* ==> 5.75 */
```

Referenciando Elementos de Arrays

- Pode-se utilizar ponteiros e colchetes:

```
float m[] = { 1.0, 3.0, 5.75, 2.345 };  
float *pf;  pf = &m[2];
```

```
cout << pf[1]; /* ==> 2.345 */
```

- Note que o valor entre colchetes é o deslocamento a ser considerado a partir do endereço de referencia
 - pf[n] => indica enésimo elemento a partir de pf

Vamos ao código C++

- Crie um programa C++;
- Parte I:
 - No programa principal defina um array (vetor) de 5 posições;
 - Inicialize os elementos desse vetor com a seguinte regra:
 - `vetor[i] = i + 2;`
 - Utilize a estrutura *for*;
 - Crie uma estrutura de repetição (for) para imprimir esses 5 elementos do vetor;
 - Teste o seu programa.
- Parte II:
 - Crie uma função chamada `init_vetor` para iniciar esse vetor:
 - O código criado para inicializar o vetor deve ser movido para essa função;
 - Crie uma função chamada `list_vetor` para listar os elementos desse vetor:
 - O código criado para imprimir os elementos desse vetor deve ser movido para essa função;
 - Teste o seu programa.
- Analise e reflita sobre as alterações.

Vamos ao código C++

```
#include <iostream>
using namespace std;
void init_vetor(float *v, int vsize){
    for (int i = 0; i < vsize; i++){
        v[i] = i + 2;
    }
}
void list_vetor(float *v, int vsize){
    for (int i = 0; i < vsize; i++){
        cout << "O elemento " << i << " do vetor e: " << v[i] << endl;
    }
}
int main(){
    int n = 5;
    float vet[n];
    cout << "o tamanho do vetor eh: " << sizeof(vet) / sizeof(vet[0]) << endl;
    init_vetor(vet, n);
    list_vetor(vet, n);
}
```

Ponteiros e Strings

- Strings são arrays de caracteres e podem ser acessadas usando char *;

```
int main(){  
    char str[] = "abcdef", *pc;  
    for (pc = str; *pc != '\0'; pc++)  
        cout << *pc;  
}
```

Resultado:

➤ abcdef

- O incremento de pc o posiciona sobre o próximo caractere (byte a byte).

Ponteiros e String

- Com base na função StrCopyC abaixo, você deverá construir um programa em C++ com duas variáveis chamadas frase1 e frase2. A variável frase1 deverá conter uma frase qualquer e deverá ser copiada para a variável frase2 utilizando a função abaixo:

```
void StrCopyC(char *destino, char *origem){  
    while (*origem){  
        *destino = *origem;  
        origem++;  
        destino++;  
    }  
    *destino = '\\0';  
}
```

Ponteiros e String

```
#include <iostream>
using namespace std;

void StrCopyC(char *destino, char *origem){ // a função tem como argumentos ponteiros
    while (*origem){
        *destino = *origem;
        origem++;
        destino++;
    }
    *destino = '\0'; //indica final da string
}

int main(){
    char frase1[50] = "Universidade Federal da Bahia", frase2[50]="";
    StrCopyC(&frase2[0], &frase1[0]); // envia o primeiro endereço da string
    cout << frase1 << endl;           // altere o programa para enviar o n-ésimo endereço de frase1
    cout << frase2 << endl;           // analise o resultado
}
```

Exercício 20

Faça um programa que leia dois valores inteiros e chame uma função que receba estes 2 valores de entrada e retorne o maior valor na primeira variável e o menor valor na segunda variável. Escreva o conteúdo das 2 variáveis na tela.

Exercício 20

```
#include <iostream>
using namespace std;
void maior_menor(int *n1, int *n2){
    int temp;
    if(*n1 < *n2){
        temp = *n1;
        *n1 = *n2;
        *n2 = temp; }
}

int main(){
    int num1, num2;
    cout << "Informe o primeiro numero: " << endl;
    cin >> num1;
    cout << "Informe o segundo numero: " << endl;
    cin >> num2;
    maior_menor(&num1, &num2);
    cout << "O maior numero e: " << num1 << endl;
    cout << "O segundo numero e: " << num2 << endl;
    return 0;
}
```



Alocação Dinâmica de Memória

Contexto

- Faça um programa para cadastrar o preço de N produtos, em que N é um valor informado pelo usuário:

```
int N, i;  
double produtos[N];
```

Errado! Não sabemos o valor de N

```
int N, i;
```

```
scanf ("%d", &N)
```

```
double produtos[N];
```

Funciona, mas não é o mais indicado

- Sempre que escrevemos um programa, é preciso reservar espaço para as informações que serão processadas;
- Para isso utilizamos as variáveis:
 - Uma variável é uma posição de memória que armazena uma informação que pode ser modificada pelo programa.
 - Ela deve ser definida antes de ser usada.
- Quanto de memória um programa irá precisar?

Alocação Dinâmica

- Permite ao programador alocar memória para variáveis quando o programa está em execução e não apenas quando se está escrevendo o programa:
 - Quantidade de memória alocada sob demanda, ou seja, quando o programa precisa;
 - Menor desperdício de memória:
 - Espaço é reservado até liberação explícita;
 - Depois de liberado, estará disponível para outros usos e não poderá mais ser acessado;
 - Espaço alocado e não liberado explicitamente é automaticamente liberado ao final da execução.

Alocação de Memória

Memória		
posição	variável	conteúdo
119		
120		
121	int *p	NULL
122		
123		
124		
125		
126		
127		
128		

**Alocando 5
posições de
memória em int *p**



Memória		
posição	variável	conteúdo
119		
120		
121	int *p	123
122		
123	p[0]	11
124	p[1]	25
125	p[2]	32
126	p[3]	44
127	p[4]	52
128		

Alocação Dinâmica de Memória

- A linguagem C ANSI usa apenas 4 funções para o sistema de alocação dinâmica de memória:
 - malloc;
 - calloc;
 - realloc;
 - free.
- Disponíveis:
 - <stdlib.h>
 - <cstdlib>

Alocação Dinâmica de Memória - malloc

- A função malloc() serve para alocar memória e tem o seguinte protótipo:
 - void *malloc (unsigned int num);
- Ou seja:
 - Dado um número de bytes que queremos alocar (num);
 - Aloca na memória e retorna um ponteiro (void*) para o primeiro byte alocada
- Se não houver memória suficiente para alocar a memória requisitada a função malloc() retorna um ponteiro nulo (NULL);

Alocação Dinâmica de Memória - malloc

- O ponteiro **void*** pode ser atribuído a qualquer tipo de ponteiro via *type cast*. Se não houver memória suficiente para alocar a memória requisitada a função **malloc()** retorna um ponteiro nulo:

```
void *malloc (unsigned int num);
```

Operador sizeof

- Operador em tempo de compilação que retorna o tamanho em **bytes**, da variável ou especificador de tipo, em parênteses, que ele precede (int, float, char, ...);

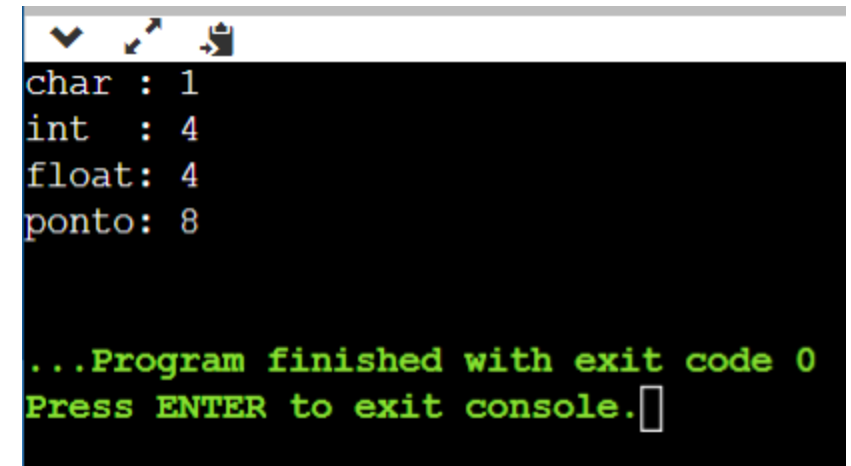
- Exemplo:

```
#include <iostream>
#include <stdlib.h>

using namespace std;

struct ponto{ int x, y;};

int main(){
    cout << "char : " << sizeof(char) << endl;
    cout << "int : " << sizeof(int) << endl;
    cout << "float: " << sizeof(float) << endl;
    cout << "ponto: " << sizeof(ponto) << endl;
    return 0;
}
```

A screenshot of a console window with a black background and white text. The output shows the sizes of various data types: char is 1, int is 4, float is 4, and ponto is 8. At the bottom, a green message states '...Program finished with exit code 0' and 'Press ENTER to exit console.' followed by a cursor.

```
char : 1
int : 4
float: 4
ponto: 8

...Program finished with exit code 0
Press ENTER to exit console.
```

Alocação Dinâmica de Memória - malloc

- Alocar 1000 bytes de memória livre.

```
char *p;  
p = (char *) malloc(1000);
```

- Alocar espaço para 50 inteiros:

```
int *p;  
p = (int *) malloc(50*sizeof(int));
```

- **cast:**
 - Utilizado para forçar uma expressão a ser de um determinado tipo;
 - Formato: (tipo) expressão;

```
int x = 100;  
float f = (float) x/2;
```


Exemplo usando malloc

```
#include <iostream>
using namespace std;
#include <stdlib.h> /* Para usar malloc() */

int main (){
    int *p;
    int a;
    int i;

    cin >> a;
    p = (int*) malloc (a*sizeof(int)); /* Aloca a números inteiros */

    if (!p){ /* ou p == NULL */
        cout << "*** Erro: Memoria Insuficiente ***" << endl;
        exit(0);
    }

    for (i=0; i<a ; i++) /*p pode ser tratado como um vetor com a posições */
        p[i] = i*a;
        cout << p[i] << endl;
    return 0;
}
```

Alocação Dinâmica de Memória - calloc

- A função `calloc()` também serve para alocar memória, mas possui um protótipo um pouco diferente:
 - `void *calloc (unsigned int num, unsigned int size);`
- Aloca memória suficiente para um vetor de **num** objetos de tamanho **size**;
- Retorna um ponteiro **void*** para o primeiro **byte** alocado;
- Retorna **NULL** se não houver memória suficiente;
- Basicamente, a função **calloc()** faz o mesmo que a função **malloc()**. A **diferença** é que agora passamos a **quantidade de posições** a serem alocadas e o **tamanho do tipo de dado** alocado como parâmetros distintos da função.

Exemplo usando calloc

```
#include <iostream>
using namespace std;
#include <stdlib.h> /* Para usar malloc() */

int main (){
    int *p;
    int a;
    int i;

    cin >> a;
    p = (int*) calloc (a, sizeof (int)); /* Aloca a números inteiros */

    if (!p){
        cout << "*** Erro: Memoria Insuficiente ***" << endl;
        exit(0);
    }

    for (i=0; i<a ; i++) /*p pode ser tratado como um vetor com a posições */
        p[i] = i*a;
        cout << p[i] << endl;
    return 0;
}
```

Alocação Dinâmica de Memória - realloc

- A função `realloc()` serve para realocar memória e tem o seguinte protótipo:
 - `void *realloc (void *ptr, unsigned int num);`
- A função modifica o tamanho da memória previamente alocada apontada por `*ptr` para aquele especificado por `num`
- O valor de `num` pode ser maior ou menor que o original
- Se não houver memória suficiente para a realocação, um ponteiro nulo é devolvido e o bloco original é deixado inalterado.

Exemplo usando realloc

```
#include <iostream>
using namespace std;
#include <stdlib.h> /* Para usar malloc() */
int main (){
    int *p; int a=30; int i;
    p = (int*) malloc (a*sizeof(int)); /* Aloca a números inteiros */
    if (!p){
        cout << "*** Erro: Memoria Insuficiente ***" << endl;
        exit(0);
    }
    for (i=0; i<a ; i++) /*p pode ser tratado como um vetor com a posições */
        p[i] = i*a;
    /* O tamanho de p deve ser modificado, por algum motivo ... */
    a = 100;
    p = realloc (p, a*sizeof(int));
    for (i=0; i<a ; i++)/* p pode ser tratado como um vetor com a posições */
        p[i] = a*i;
    return 0;
}
```

Alocação Dinâmica de Memória - realloc

- Observações sobre realloc()
 - Se `*ptr` for nulo, aloca `num` bytes e devolve um ponteiro (igual malloc);
 - se `num` é zero, a memória apontada por `*ptr` é liberada (igual free).
 - Se não houver memória suficiente para a alocação, um ponteiro nulo é devolvido e o bloco original é deixado inalterado.

Alocação Dinâmica de Memória - free

- Diferente das variáveis definidas durante a escrita do programa, as variáveis alocadas dinamicamente não são liberadas automaticamente pelo programa.
- Quando alocamos memória dinamicamente é necessário que nós a liberemos quando ela não for mais necessária.
- Para isto existe a função `free()` cujo protótipo é:
 - `void free (void* p);`

Alocação Dinâmica de Memória - free

- Assim, para liberar a memória, basta passar como parâmetro para a função `free()` o ponteiro que aponta para o início da memória a ser desalocada.
- Como o programa sabe quantos bytes devem ser liberados?
 - Quando se aloca a memória, o programa guarda o número de bytes alocados numa "tabela de alocação" interna.

Exemplo usando free

```
#include <iostream>
using namespace std;
#include <stdlib.h> /* Para usar malloc() */

int main (){
    int *p; int a;  int i;
    cin >> a;
    p = (int*) malloc (a*sizeof(int)); /* Aloca a números inteiros */

    if (!p){
        cout << "*** Erro: Memoria Insuficiente ***" << endl;
        exit(0);
    }

    ...
    free(p);
    ...
    return 0;
}
```

Exercício 21

- Faça um programa que pergunte ao usuário quantos valores ele deseja armazenar em um vetor de double, depois use a função MALLOC para reservar (alocar) o espaço de memória de acordo com o especificado pelo usuário. Esse vetor deve ter um tamanho maior ou igual a 10 elementos. Use este vetor dinâmico como um vetor comum, atribuindo aos 10 primeiros elementos do vetor os valores dos respectivos índices + 100. Exiba na tela os valores armazenados nos 10 primeiros elementos do vetor.

Resumo

Função	Finalidade	Sintaxe	Exemplo
malloc	Alocar memória	void *malloc (unsigned int num)	char *p; p = (char *) malloc(100) int *p; p = (int *) malloc(50*sizeof(int));
calloc	Alocar memória	void *calloc (unsigned int num, unsigned int size)	int *p; p = (int *) calloc(50,sizeof(int));
realloc	Realocar memória	void *realloc (void *ptr, unsigned int num)	int *p; p = (int *) calloc(50,sizeof(int)); p = realloc(p, 10*sizeof(int));
free	Liberar memória	free(p)	int *p; p = (int *) calloc(50,sizeof(int)); p = realloc(p, 10*sizeof(int)); free(p);

Exercício 22

- Faça um programa que leia uma quantidade qualquer de números armazenando-os na memória e pare a leitura quando o usuário entrar um número negativo. Em seguida, imprima o vetor lido. Use a função REALLOC