

Alocação Dinâmica e Ponteiros

Eduardo Piveta

Introdução

- Ponteiros e referências
- Alocação estática vs. dinâmica

Ponteiros em C

- Operadores usados para ponteiros:
 - * (define que um tipo é um ponteiro, conteúdo de)
 - & (endereço de)
 - > (equivalente a (*v).algo)

Declarando ponteiros em C

- Ponteiro para um inteiro:
 - `int* x; // ou int *x;`
- Ponteiro para um caractere:
 - `char* c;`
- Ponteiro para uma pessoa:
 - `Pessoa* p;`
- Ponteiro para uma lista:
 - `Lista* l;`
- ..

Alocando memória em C

- Funções:
 - malloc: aloca memória
 - free: libera memória
- Funções Auxiliares:
 - sizeof: determina o tamanho de um tipo

Alocando memória em C

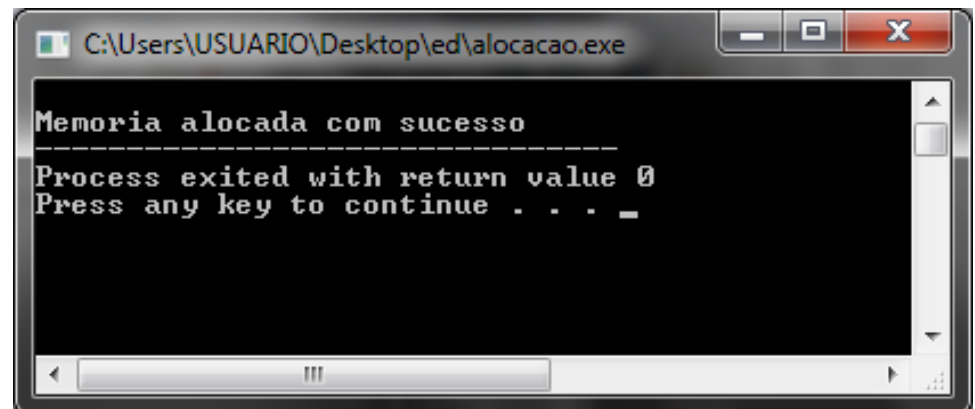
- Alocando memória para um inteiro:
 - `int* x = (int*) malloc(sizeof(int));`
- Alocando memória para um caractere:
 - `char* c = (char*) malloc(sizeof(char));`
- Alocando memória para uma pessoa:
 - `Pessoa* p = (Pessoa*) malloc(sizeof(Pessoa));`
- Alocando memória para uma lista:
 - `Lista* l = (Lista*) malloc(sizeof(Lista));`
- ..

Liberando memória em C

- Liberando memória para um inteiro:
 - `free(x);`
- Liberando memória para um caractere:
 - `free(c);`
- Liberando memória para uma pessoa:
 - `free(p);`
- Liberando memória para uma lista:
 - `free(l);`
- ..

Exemplo 1

```
alocacao.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct pessoa{
5      int idade;
6      char nome[20];
7  } Pessoa;
8
9  int main(int argc, char *argv[]) {
10     Pessoa* p = (Pessoa*) malloc(sizeof(Pessoa));
11     if (p!=NULL){
12         printf("\nMemoria alocada com sucesso");
13         // Realiza operacoes
14     }
15     free(p);
16     return 0;
17 }
```



Leitura e escrita

- O acesso pode ser feito de duas formas:
 - Através do operador * ou
 - Através do operador ->

Leitura e Escrita (operador *)

alocacao.c alocacaoLeituraEscrita1.cpp

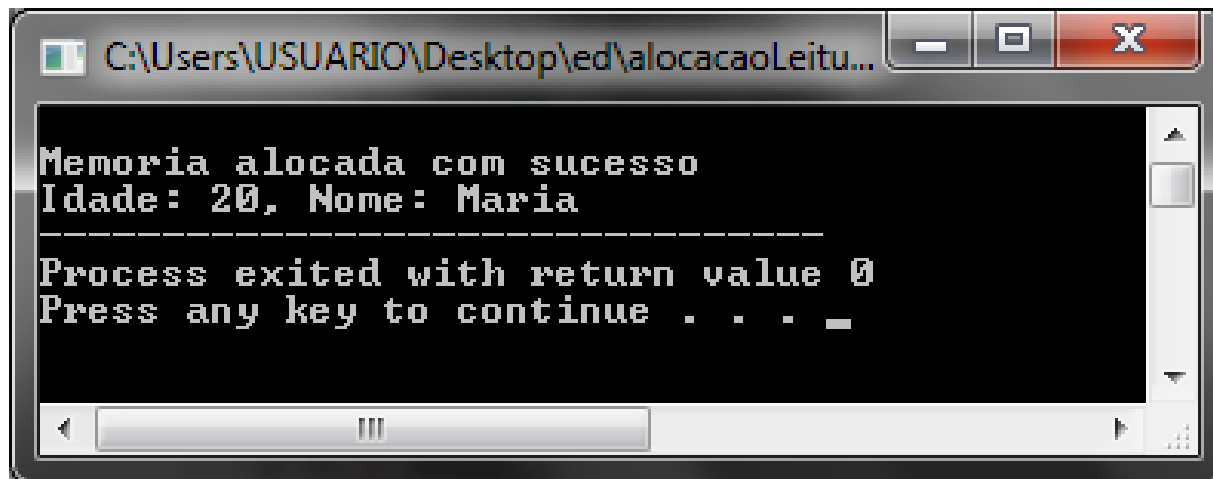
```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  typedef struct pessoa{
6      int idade;
7      char nome[20];
8  } Pessoa;
9
10 int main(int argc, char *argv[]) {
11     Pessoa* p = (Pessoa*) malloc(sizeof(Pessoa));
12     if (p!=NULL){
13         printf("\nMemoria alocada com sucesso");
14         (*p).idade = 20;
15         strcpy((*p).nome, "Maria");
16         printf("\nIdade: %d, Nome: %s", (*p).idade, (*p).nome);
17     }
18     free(p);
19     return 0;
20 }
```

Leitura e Escrita (operador *)

alocacao.c alocacaoLeituraEscrita1.cpp

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  typedef struct pessoa{
6      int idade;
7      char nome[20];
8  } Pessoa;
9
10 int main(int argc, char *argv[]) {
11     Pessoa* p = (Pessoa*) malloc(sizeof(Pessoa));
12     if (p!=NULL){
13         printf("\nMemoria alocada com sucesso");
14         (*p).idade = 20;
15         strcpy((*p).nome, "Maria");
16         printf("\nIdade: %d, Nome: %s", (*p).idade, (*p).nome);
17     }
18     free(p);
19     return 0;
20 }
```

Execução



A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\USUARIO\Desktop\ed\alocacaoLeitu...". The window has standard Windows window controls (minimize, maximize, close). The command prompt displays the following text:

```
Memoria alocada com sucesso  
Idade: 20, Nome: Maria  
-----  
Process exited with return value 0  
Press any key to continue . . . _
```

The text is displayed in a monospaced font on a black background. A horizontal line separates the program output from the exit message. The prompt is currently at the end of the last line.

Leitura e Escrita (operador ->)

alocacao.c alocaoLeituraEscrita2.cpp

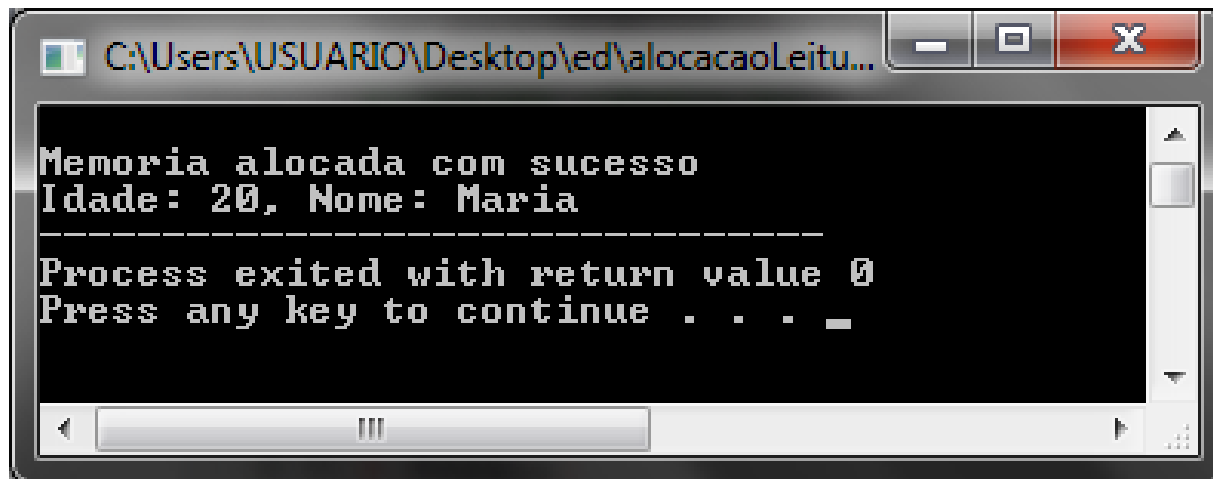
```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  typedef struct pessoa{
6      int idade;
7      char nome[20];
8  } Pessoa;
9
10 int main(int argc, char *argv[]) {
11     Pessoa* p = (Pessoa*) malloc(sizeof(Pessoa));
12     if (p!=NULL) {
13         printf("\nMemoria alocada com sucesso");
14         p->idade = 20;
15         strcpy(p->nome, "Maria");
16         printf("\nIdade: %d, Nome: %s", p->idade, p->nome);
17     }
18     free(p);
19     return 0;
20 }
```

Leitura e Escrita (operador ->)

alocacao.c alocacaoLeituraEscrita2.cpp

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  typedef struct pessoa{
6      int idade;
7      char nome[20];
8  } Pessoa;
9
10 int main(int argc, char *argv[]) {
11     Pessoa* p = (Pessoa*) malloc(sizeof(Pessoa));
12     if (p!=NULL){
13         printf("\nMemoria alocada com sucesso");
14         p->idade = 20;
15         strcpy(p->nome,"Maria");
16         printf("\nIdade: %d, Nome: %s", p->idade, p->nome);
17     }
18     free(p);
19     return 0;
20 }
```

Execução



A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\USUARIO\Desktop\ed\alocacaoLeitu...". The window has standard Windows window controls (minimize, maximize, close). The command prompt displays the following text:

```
Memoria alocada com sucesso  
Idade: 20, Nome: Maria  
-----  
Process exited with return value 0  
Press any key to continue . . . _
```

The text is displayed in a monospaced font on a black background. A horizontal line separates the program output from the exit message. The prompt is currently at the end of the last line.

Passagem por referência

- Passagem por valor vs. Passagem por referência

Passagem por referência em C

```
10 void mostrar(Pessoa* x) {  
11     printf("\nIdade: %d, Nome: %s", x->idade, x->nome);  
12 }  
13  
14 int main(int argc, char *argv[]) {  
15     Pessoa* p = (Pessoa*) malloc(sizeof(Pessoa));  
16     if (p!=NULL) {  
17         printf("\nMemoria alocada com sucesso");  
18         p->idade = 20;  
19         strcpy(p->nome, "Maria");  
20         mostrar(p);  
21     }  
22     free(p);  
23     return 0;  
24 }
```

Retornando Referências

```
14 Pessoa* criar() {  
15     return (Pessoa*) malloc(sizeof(Pessoa));  
16 }  
17  
18 int main(int argc, char *argv[]) {  
19     Pessoa* p = criar();  
20     if (p!=NULL) {  
21         printf("\nMemoria alocada com sucesso");  
22         p->idade = 20;  
23         strcpy(p->nome, "Maria");  
24         mostrar(p);  
25     }  
26     free(p);  
27     return 0;  
28 }
```