FAGAMMON
FACULDADE PRESBITERIANA GAMMON
30 ANOS

Algoritmos e Estrutura de Dados II

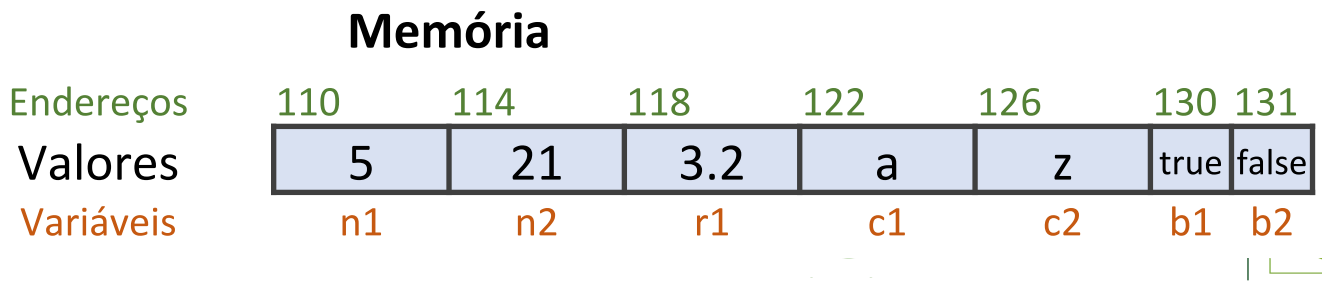
Aula 9

Ponteiros e Alocação Dinâmica

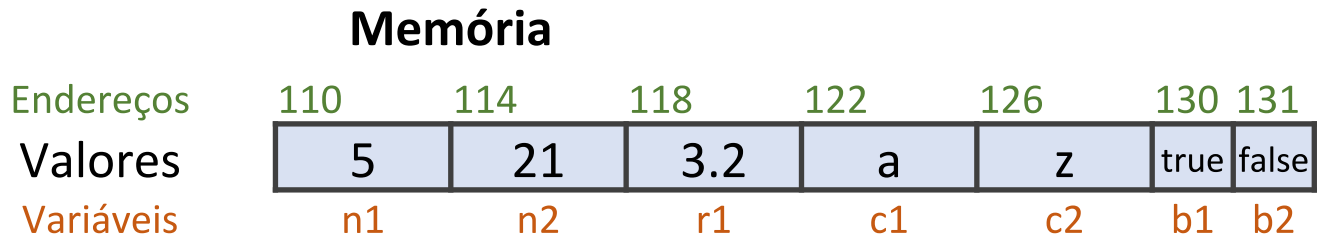
Claudiane Maria Oliveira
claudianeo@gmail.com

Introdução

- Variáveis
  - São de um Tipo.
  - Possuem um Identificador (nome).
  - Ocupam uma Região de Memória.
- Exemplos
  - n1 e n2: inteiro int n1, n2;
  - r1: real float r1;
  - c1 e c2: caractere char c1, c2;
  - b1 e b2: boolean boolean b1, b2;



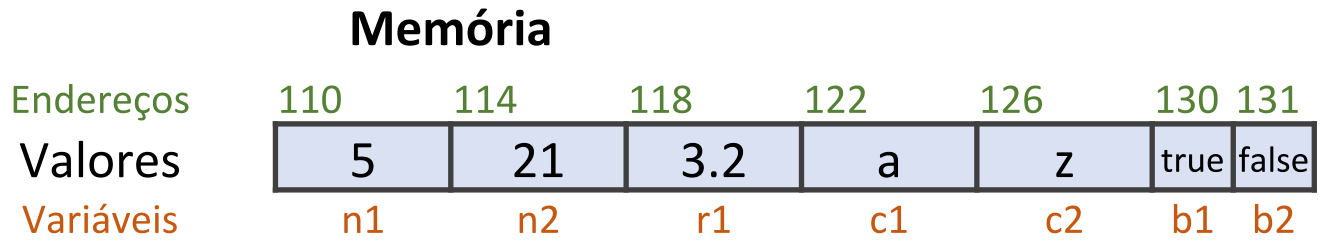
Introdução



- Uma variável é armazenada em um local da memória.
- E cada posição de memória tem um endereço.

- **E ponteiro?**

Introdução



- Uma variável é armazenada em um local da memória.
- E cada posição de memória tem um endereço.

- **E ponteiro?**
  - É uma variável que armazena um endereço de memória.

Ponteiros

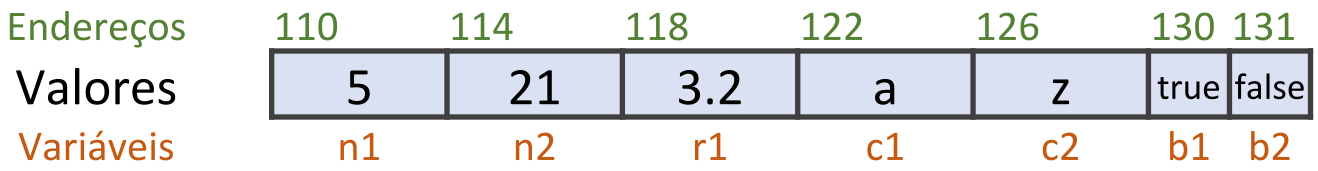
➢ **Se um ponteiro** é uma variável, ele tem um tipo, um identificador e ocupa uma região de memória.

```
• int n1 = 5;
• int n2 = 21;
• float r1 = 3.2;
• Char c1 = ‘a’;
• Char c2 = ‘z’;
• Bool b1 = true;
• Bool b2 = false;

• int *p1;
```

Declaramos um ponteiro com um tipo, o símbolo * e o nome da variável.

Nesse exemplo p1 é um ponteiro capaz de guardar um endereço de memória de uma variável inteiro.

Memória



Operador & - Endereço de Variáveis

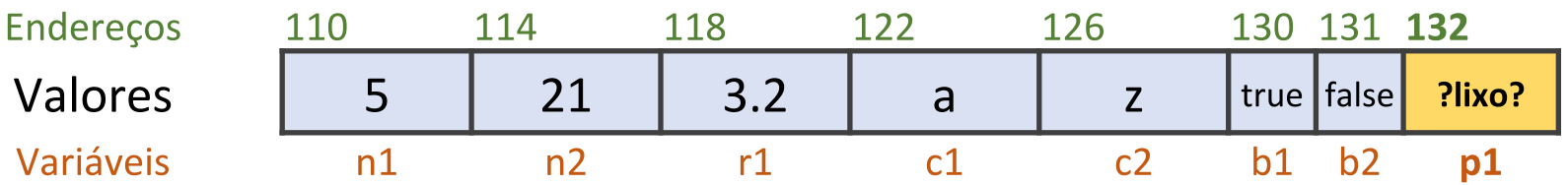
- **Um ponteiro** serve para guardar o endereço de uma outra variável.

- Em C++, o operador **&** nos permite pegar o endereço de uma variável.

- Portanto se:
  - n é uma variável do tipo inteiro.
  - &n representa o endereço de n

Operador & - Endereço de Variáveis

Memória



&n1	110
&n2	114
&r1	118
&c1	122
&c2	126
&b1	130
&b2	131

Operador & - Endereço de Variáveis

➢ **Na prática**

```
 1  #include <iostream>
 2
 3  using namespace std;
 4
 5  int main()
 6  {
 7      int n = 10;
 8
 9      cout << "O valor de n eh:\t"    <<  n << endl;
10      cout << "O endereco de n eh:\t" << &n << endl;
11  }
```

```
O valor de n eh:        10
O endereco de n eh:     0x61ff1c
```

Operador & - Endereço de Variáveis

➢ Na prática

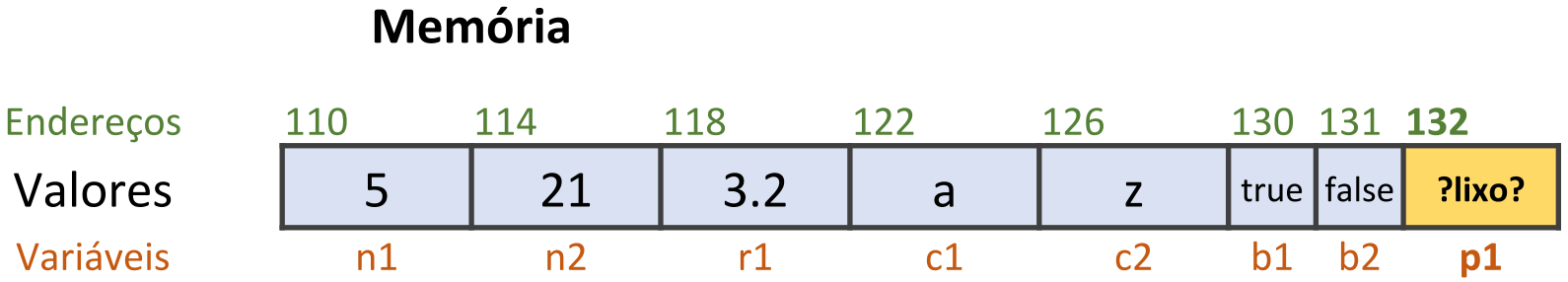
```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int n = 10;
8
9      cout << "O valor de n eh:\t"    <<  n << endl;
10     cout << "O endereco de n eh:\t" << &n << endl;
11 }
```

```
O valor de n eh:        10
O endereco de n eh:     0x61ff1c
```

Usando o operador & exibimos o endereço de n.

Os endereços de memória são geralmente representados em valores hexadecimais (começam com 0x).

Usando Ponteiros

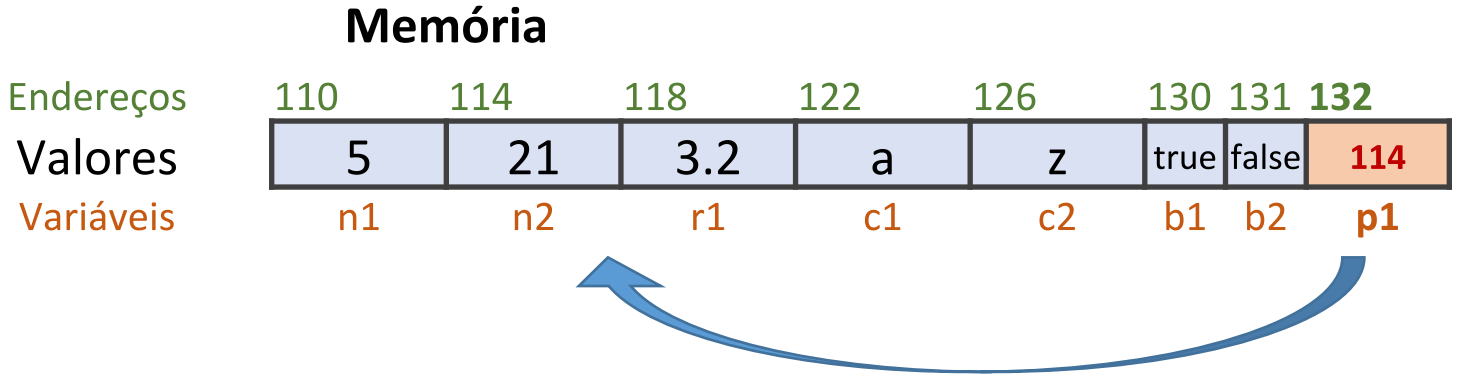


- & permite pegar o endereço de uma variável
- Ponteiro guarda o endereço de uma variável

- O comando a seguir é permitido?

```
int *p1 = &n2;
```

Usando Ponteiros



- O comando a seguir é permitido?

```
int *p1 = &n2;
```

- Dizemos que p1 agora aponta para n2

Tipos de Ponteiros

- Temos ponteiro para armazenar endereços de memória de variáveis de todos os tipos

int *x; // x é uma variável capaz de armazenar o endereço de um int.

float *y; // y é uma variável capaz de armazenar o endereço de um float.

bool *z; // z é uma variável capaz de armazenar o endereço de um bool.

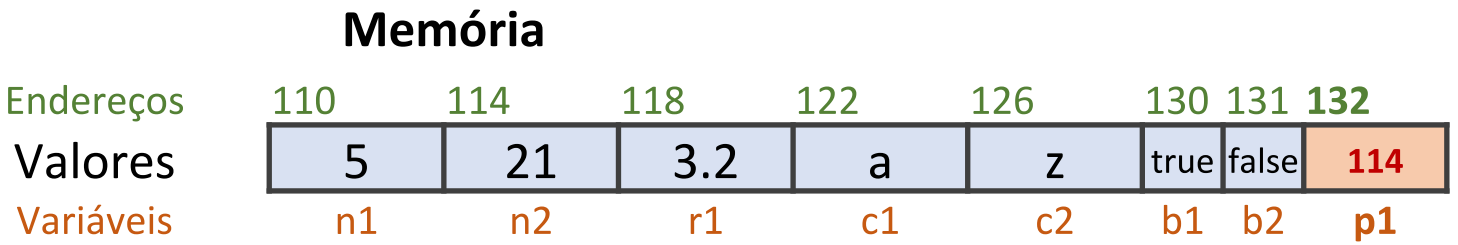
Inicializando Ponteiros

- Podemos inicializar o ponteiro na declaração ou depois, como fazemos com qualquer variável.

```
int numero = 10;
int *pNumero = &numero;
```

```
int numero = 10;
int *pNumero;

// alguns codigos...

pNumero = &numero;
```

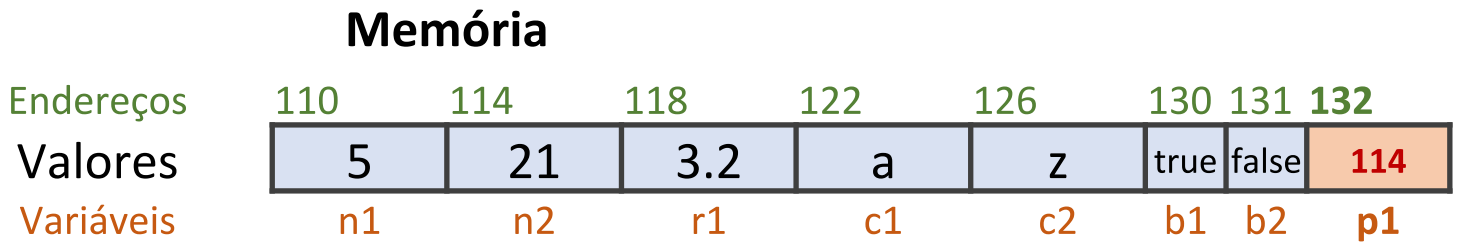
Exemplo de Ponteiros



- O que aparecerá na tela se executar os seguintes comandos?

```
cout <<  n2 << endl;

cout << &n2 << endl;

cout <<  p1 << endl;

cout << &p1 << endl;
```

Exemplo de Ponteiros



- O que aparecerá na tela se executar os seguintes comandos?

```
cout <<  n2 << endl;

cout << &n2 << endl;

cout <<  p1 << endl;

cout << &p1 << endl;
```

```
21

114

114

132
```

Operador de indireção *

- Usamos o operador de indireção * para acessar a variável apontada de forma indireta.

```
int n = 10;
int *p = &n;
cout << "Valor apontado por p: " << *p << endl;
```

```
Valor apontado por p: 10
```

Ao usar ***p** estamos usando o endereço guardado em p, para buscar o valor neste endereço (ou seja, o valor de n).

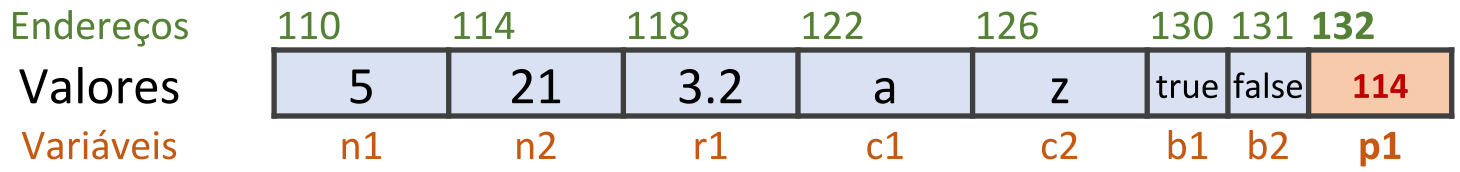
Operador de indireção *

- Voltando ao exemplo
  - O que é exibido agora usando o ***p1** ?

```
cout <<  n2 << endl;

cout << &n2 << endl;

cout <<  p1 << endl;

cout << &p1 << endl;

cout << *p1 << endl;
```

```
21

114

114

132
```

Memória



Operador de indireção *

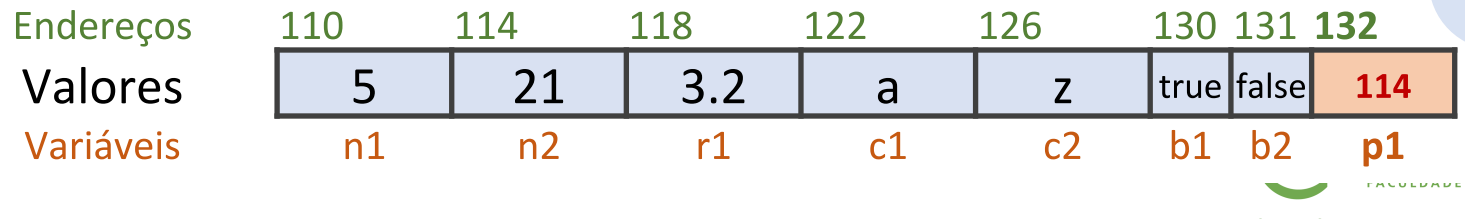
➢ O que é exibido agora usando o ***p1*** ?

```
cout <<  n2 << endl;

cout << &n2 << endl;

cout <<  p1 << endl;

cout << &p1 << endl;

cout << *p1 << endl;
```

```
21

114

114

132

21
```

Ou seja, usar ***p1*** significa acessar indiretamente a variável **n2** apontada por ele.

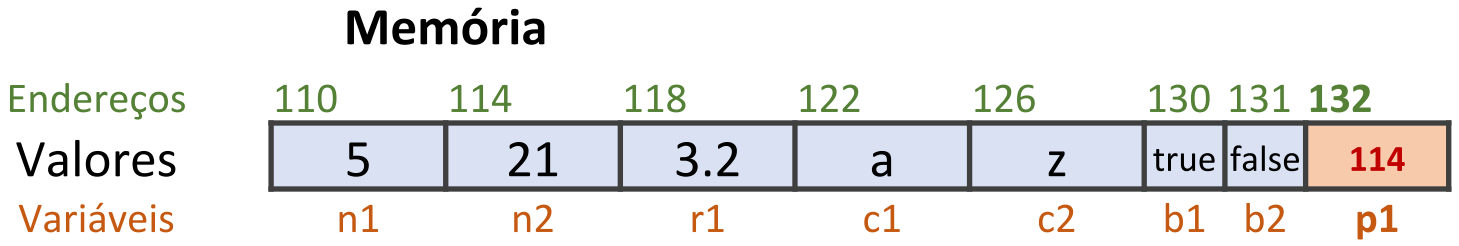
Memória



Operador de indireção *

- Com o operador de indireção * além de acessar uma variável indiretamente, você pode também alterar seu valor.
- Se executarmos o comando:

```
*p1 = 9;
// no exemplo abaixo, estaremos alterando o valor de n2.
```



Repare que ao usar *p1, não estamos alterando o valor de p1 (o endereço 114), mas sim o valor da variável apontada por esse endereço.

Exemplos do operador de indireção *


```
int n = 10;
int *p = &n;

cout << "Valor de n: " << n << endl;
cout << "Valor apontado por p: " << *p << endl;

*p = 18;

cout << "Valor de n: " << n << endl;
cout << "Valor apontado por p: " << *p << endl;
```

```
Valor de n: 10
Valor apontado por p: 10
Valor de n: 18
Valor apontado por p: 18
```

Exemplos de Ponteiros

O que é impresso na tela.

```
float f = 2.1;
float *x = &f;

cout << *x << endl;
cout << x << endl;
cout << f << endl;
cout << &f << endl;
```

```
2.1
0x61ff18
2.1
0x61ff18
```

```
int n = 2;
int b = 4;
int *q = &n;
cout << n << ", " << b << ", " << *q << endl;
q = &b;
cout << n << ", " << b << ", " << *q << endl;
*q = n;
cout << n << ", " << b << ", " << *q << endl;
```

```
2, 4, 2
2, 4, 4
2, 2, 2
```

```
int n1 = 5;
int *p = &n1;
int n2 = 20;
int *c = &n2;
*c = n1 + n2 + *p;
cout << c << endl;
cout << n2 << endl;
```

```
0x61ff10
30
```

Exemplos de Ponteiros

O que faz o código a seguir?

```
int a, b, c;
cin >> a >> b;

int *p;
int *q;
p = &a;
q = &b;
c = *p + *q;

cout << c << endl;
```

Referência e derreferenciação


```
int *p1 = &n2;
```

- Além de dizer que p1 aponta para n2, podemos dizer que p1 **faz referência** a n2.

- Por isso, a utilização do operador de indireção * também é chamada de "derreferenciação".

```
cout << *p1 << endl;
```

Importância de Ponteiros

- Ponteiros são muito utilizados por diversos motivos:
  - Por ser mais eficiente (velocidade de acesso) do que outras formas.
  - Para alocação dinâmica de memória (assunto que veremos na próxima aula).
  - São úteis na criação de estruturas de dados (listas, árvores, etc.).
  - Para quando precisamos de acesso direto ao hardware.

Cuidados com Ponteiros

- O uso de ponteiro exige alguns cuidados!
  - Qual é o problema de cada situação abaixo:

```
int *x;
*x = 10;
```

```
int *x;
x = 10;
```

```
int n;
int *x = &n;
int *y = *x;
```

Cuidados com Ponteiros

- O uso de ponteiro exige alguns cuidados!
  - Qual é o problema de cada situação abaixo:

```
int *x;
*x = 10;
```

O **ponteiro** x **não** foi **inicializado**, portanto ele guarda um endereço lixo. Ou seja, estamos tentando guardar o valor 10 em uma posição desconhecida da memória.

```
int *x;
x = 10;
```

```
int n;
int *x = &n;
int *y = *x;
```

Cuidados com Ponteiros

- O uso de ponteiro exige alguns cuidados!
  - Qual é o problema de cada situação abaixo:

```
int *x;
*x = 10;
```

O **ponteiro** x **não** foi **inicializado**, portanto ele guarda um endereço lixo. Ou seja, estamos tentando guardar o valor 10 em uma posição desconhecida da memória.

```
int *x;
x = 10;
```

Estamos tentando **alterar** o **endereço** para onde o **ponteiro** x aponta. Mas o que tem no endereço 10 do computador?

```
int n;
int *x = &n;
int *y = *x;
```

Cuidados com Ponteiros

- O uso de ponteiro exige alguns cuidados!
  - Qual é o problema de cada situação abaixo:

```
int *x;
*x = 10;
```

O **ponteiro** x **não** foi **inicializado**, portanto ele guarda um endereço lixo. Ou seja, estamos tentando guardar o valor 10 em uma posição desconhecida da memória.

```
int *x;
x = 10;
```

Estamos tentando **alterar** o **endereço** para onde o **ponteiro** x aponta. Mas o que tem no endereço 10 do computador?

```
int n;
int *x = &n;
int *y = *x;
```

É o mesmo problema do item anterior. Estamos tentando colocar como endereço do ponteiro y, o valor apontado pelo ponteiro x.

Detalhe: cuidado para a diferença de *y na declaração e *y em

Cuidados com Ponteiros

- Qual é a diferença entre os códigos abaixo?

```
int *x = 10;
```

No primeiro código *x significa declarar um ponteiro para inteiro. Ao atribuir 10, estamos tentando fazer o ponteiro apontar para o endereço de memória 10.

```
int *x;
*x = 10;
```

Já no segundo exemplo, após declarar o ponteiro x, estamos usando indireção, *x, para tentar alterar o conteúdo do endereço para onde x aponta.

Cuidados com Ponteiros

- O que exibe na tela o código abaixo?

```
int n = 10;
int *p = &n;

cout << *p *n << endl;
```

Cuidados com Ponteiros

- O que exibe na tela o código abaixo?

```
int n = 10;
int *p = &n;

cout << *p *n << endl;
```

```
100
```

Cuidado para não confundir os usos do *****. Ele pode ser o **operador de indireção** ou o **operador de multiplicação**.

O compilador considera que é um ou outro dependendo da variável em questão.

No código acima o ***** antes do **p** é identificado como indireção porque **p** é um ponteiro. Já o ***** antes do **n** é identificado como multiplicação porque **n** é uma variável comum (escalar).

Exercícios

1) Escreva um programa que declare duas variáveis inteiras **x** e **y** e um ponteiro **p**. O ponteiro **p** deve apontar para a variável **y**. O valor de **x** deve ser solicitado ao usuário e, em seguida, o valor da variável **y** deve ser alterado de forma indireta, através do ponteiro **p**, para receber o valor de **x** ao cubo. Por fim, imprima o valor de **y**.

Exercícios

2) Aponte os erros no programa abaixo de forma que o programa faça o que é descrito nos comentários.

```
 8     int x, y;
 9     x = 10;
10     int *p;
11
12     // p aponta para x
13     p = *x;
14
15     // y recebe o valor apontado por p
16     y = p;
17
18     // exibe o valor de y, que é 10
19     cout << *y << endl;
```

Exercícios

3) Faça uma função que receba um ponteiro para um número inteiro e retorne o dobro do número.

4) Faça uma função que receba três ponteiros para números reais e retorne o endereço do maior número.

Links para as aulas

- **Aula de AED2 - Realizada no dia 31/03/2020 - 19:00**

  https://drive.google.com/file/d/15KEUJC74LiLAY9YtRlEkn-aEzJ88a9cb/view?usp=sharing

- **Aula de LAB2 - Realizada no dia 31/03/2020 - 20:55**

  https://drive.google.com/file/d/1wTHbt4dMhQ3H-M7AyfLpz4easg1f1BRQ/view?usp=sharing

FAGAMMON
FACULDADE PRESBITERIANA GAMMON
30 ANOS

Algoritmos e Estrutura de Dados II

Aula 9

Ponteiros e Alocação Dinâmica

Claudiane Maria Oliveira
claudianeo@gmail.com