ALGORITMOS E ESTRUTURA DE DADOS

Curso de Engenharia de Software
Lucas Sampaio Leite

Ponteiros

INSTITUTO FEDERAL
Pernambuco

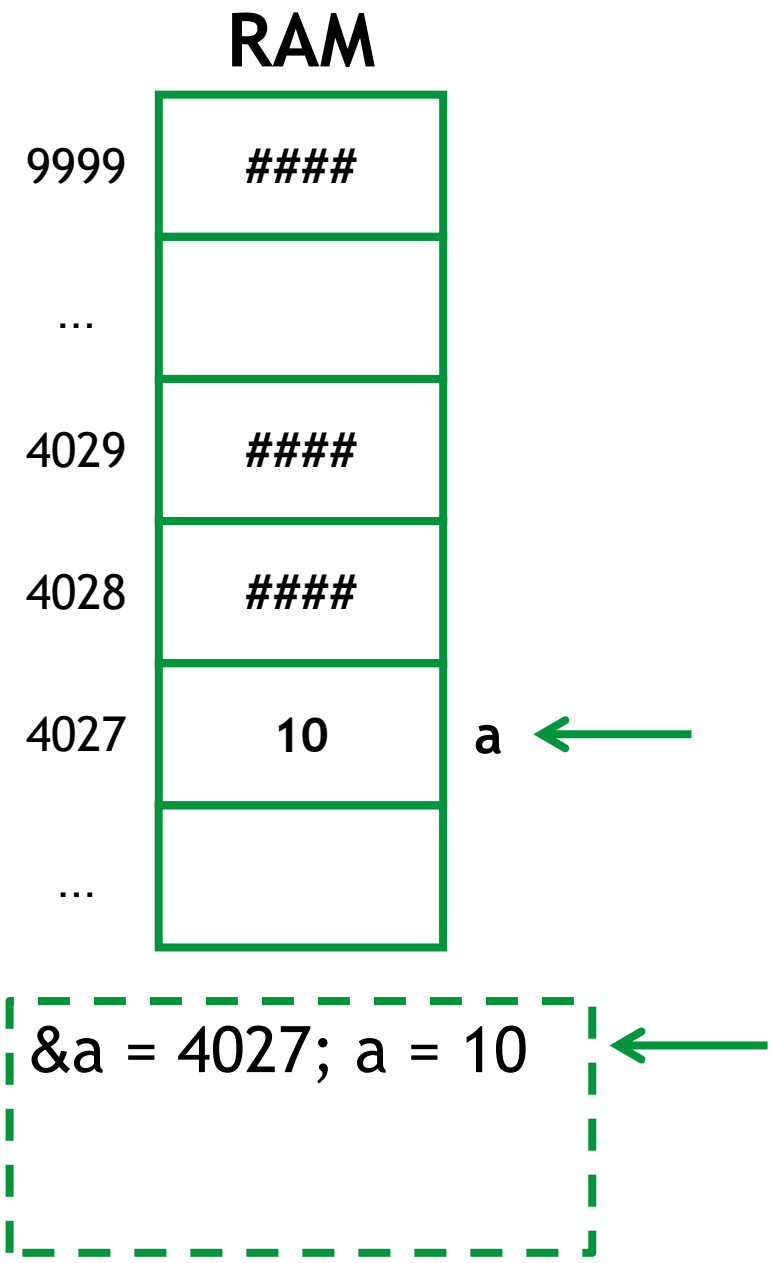
- Em linguagem C, toda variável possui quatro características fundamentais:
  - Nome: identificador usado para referenciar a variável no código;
  - Tipo: especifica o conjunto de valores possíveis e as operações permitidas;
  - Valor: conteúdo armazenado na variável em determinado momento;
  - Endereço: localização na memória onde esse valor está guardado.

Ponteiros


```c
#include <stdio.h>

int main(){
    int a = 10;
    int b, c;

    b = a;
    c = a + b;

    return 0;
}
```

Ponteiros

```c
#include <stdio.h>

int main(){
    int a = 10;
    int b, c;

    b = a;
    c = a + b;

    return 0;
}
```



INSTITUTO FEDERAL
Pernambuco

Ponteiros

- Variáveis e memória:
  - Valores podem ser alterados ao longo da execução do programa.
  - Cada variável possui um endereço fixo na memória durante a execução.
  - O conteúdo armazenado nesse endereço pode mudar, mas o endereço não se altera.

Ponteiros

INSTITUTO FEDERAL
Pernambuco

- Variáveis e memória:
  - Valores podem ser alterados ao longo da execução do programa.
  - Cada variável possui um endereço fixo na memória durante a execução.
  - O conteúdo armazenado nesse endereço pode mudar, mas o endereço não se altera.

```c
int a = 10;
int b, c;

printf("a = %d; &a = %p \n", a, &a);
printf("b = %d; &b = %p \n", b, &b);
printf("c = %d; &c = %p \n", c, &c);

b = a;
c = a + b;

printf("a = %d; &a = %p \n", a, &a);
printf("b = %d; &b = %p \n", b, &b);
printf("c = %d; &c = %p \n", c, &c);
```

Ponteiros

```c
#include <stdio.h>

int main(){

    int a = 10;
    int b, c;

    printf("a = %d; &a = %p \n", a, &a);
    printf("b = %d; &b = %p \n", b, &b);
    printf("c = %d; &c = %p \n", c, &c);

    b = a;
    c = a + b;

    printf("a = %d; &a = %p \n", a, &a);
    printf("b = %d; &b = %p \n", b, &b);
    printf("c = %d; &c = %p \n", c, &c);

    return 0;
}
```

```
a = 10; &a = 0x7fffffffd65c
b = 4096; &b = 0x7fffffffd660
c = 0; &c = 0x7fffffffd664
a = 10; &a = 0x7fffffffd65c
b = 10; &b = 0x7fffffffd660
c = 20; &c = 0x7fffffffd664
```

INSTITUTO FEDERAL
Pernambuco

Ponteiros


```
a = 10; &a = 0x7fffffffd65c
b = 4096; &b = 0x7fffffffd660
c = 0; &c = 0x7fffffffd664
a = 10; &a = 0x7fffffffd65c
b = 10; &b = 0x7fffffffd660
c = 20; &c = 0x7fffffffd664
```

```c
#include <stdio.h>

int main(){

    int a = 10;
    int b, c;

    printf("a = %d; &a = %p \n", a, &a);
    printf("b = %d; &b = %p \n", b, &b);
    printf("c = %d; &c = %p \n", c, &c);

    b = a;
    c = a + b;

    printf("a = %d; &a = %p \n", a, &a);
    printf("b = %d; &b = %p \n", b, &b);
    printf("c = %d; &c = %p \n", c, &c);

    return 0;
}
```

INSTITUTO FEDERAL
Pernambuco

Ponteiros (revisão)

As variáveis b e c são declaradas, mas não são inicializadas.

Em C, variáveis locais não inicializadas contêm qualquer valor imprevisível que já estava naquele espaço de memória. **É o famoso lixo de memória (garbage value).**

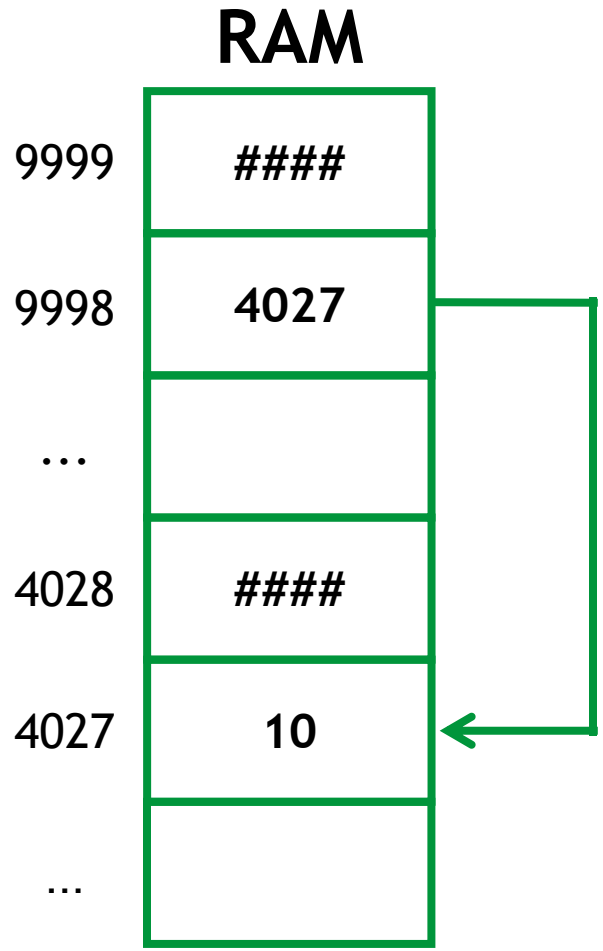
```
a = 10; &a = 0x7fffffffd65c
b = 4096; &b = 0x7fffffffd660
c = 0; &c = 0x7fffffffd664
a = 10; &a = 0x7fffffffd65c
b = 10; &b = 0x7fffffffd660
c = 20; &c = 0x7fffffffd664
```

```c
#include <stdio.h>

int main(){
                                    0;


                                = %d; &a = %p \n", a, &a);
                                = %d; &b = %p \n", b, &b);
                                = %d; &c = %p \n", c, &c);

    b = a;
    c = a + b;

    printf("a = %d; &a = %p \n", a, &a);
    printf("b = %d; &b = %p \n", b, &b);
    printf("c = %d; &c = %p \n", c, &c);

    return 0;
}
```

INSTITUTO FEDERAL
Pernambuco

Ponteiros

- O que é um ponteiro?
  - Um ponteiro é uma variável que armazena um endereço de memória.
  - Esse endereço geralmente corresponde à localização de outra variável.
  - Quando uma variável contém o endereço de outra, dizemos que ela “aponta” para essa variável.



Ponteiros

INSTITUTO FEDERAL
Pernambuco

- Um ponteiro em C é uma variável que armazena o endereço de memória de outra variável. Em outras palavras, ele “aponta” para o local onde um dado está armazenado, permitindo acessá-lo e manipulá-lo indiretamente.
- Dominar ponteiros é essencial para programar de forma eficiente em C, pois eles oferecem flexibilidade, desempenho e controle direto da memória.
- Ponteiros são amplamente utilizados em diversos contextos, como:
  - alocação dinâmica de memória (malloc, calloc, free),
  - implementação de tipos abstratos de dados (ADTs), como listas, pilhas e filas,
  - passagem eficiente de parâmetros para funções.
  - alterar variáveis dentro de funções

Ponteiros

INSTITUTO FEDERAL
Pernambuco

- Para declarar uma variável ponteiro em C, utilizamos o tipo base (ou seja, o tipo de dado para o qual o ponteiro irá apontar), seguido de um asterisco (*) e do nome da variável.
- O símbolo * é chamado de operador de indireção (ou operador de desreferência), pois permite acessar o valor armazenado no endereço ao qual o ponteiro aponta.
- A forma geral de declaração é:
  - **tipo *nome;**
  - tipo* nome;
  - tipo * nome;

Ponteiros

INSTITUTO FEDERAL
Pernambuco

- Declarando ponteiros:

```
int *pInteiro;
char *pString;
float *pFloat;

void *pGenerico;
```

O tipo do ponteiro define o tipo do dado armazenado no endereço para o qual ele aponta.

Ponteiro genérico (void*): pode armazenar o endereço de qualquer tipo de dado. No entanto, ele não pode ser desreferenciado diretamente. Para acessar o valor apontado, é necessário realizar um casting para um tipo de ponteiro específico antes do uso.

Ponteiros

INSTITUTO FEDERAL
Pernambuco

- O operador & (operador de endereço) é um operador unário que retorna o endereço de memória de uma variável.
  - Ele não tem relação com o valor armazenado na variável, apenas com a posição onde ela está guardada.

- Exemplo: `x = &valor;`
  - Essa instrução faz com que x receba o endereço da variável valor.
  - Leitura: “x recebe o endereço da variável valor”.

Ponteiros

INSTITUTO FEDERAL
Pernambuco

- O operador * (operador de indireção ou desreferência) é um operador unário que acessa o valor armazenado no endereço contido em um ponteiro.
  - Ou seja, ele permite desreferenciar um ponteiro (ler o conteúdo localizado no endereço para o qual ele aponta).
- Exemplo:

```
valor = 30;
x = &valor;
y = *x;
```

  - Essa instrução faz com que y receba o conteúdo da variável valor, que é 30.
  - Leitura: “y recebe o valor armazenado no endereço apontado por x”.

Ponteiros

INSTITUTO FEDERAL
Pernambuco

```c
int *x, valor, y;

valor = 35;
x = &valor;
y= *x;

printf("Endereço da variável 'valor'.................: %p\n", &valor);
printf("Conteúdo do ponteiro x (endereço armazenado).: %p\n", x);
printf("Endereço da variável 'x'.....................: %p\n", &x);
printf("Valor apontado por x (conteúdo de *x)........: %d\n", *x);
printf("Conteúdo da variável y.......................: %d\n", y);
```

Ponteiros

INSTITUTO FEDERAL
Pernambuco

```c
int *x, valor, y;

valor = 35;
x = &valor;
y= *x;

printf("Endereço da variável 'valor'.................: %p\n", &valor);
printf("Conteúdo do ponteiro x (endereço armazenado).: %p\n", x);
printf("Endereço da variável 'x'.....................: %p\n", &x);
printf("Valor apontado por x (conteúdo de *x).........: %d\n", *x);
printf("Conteúdo da variável y.......................: %d\n", y);
```

```
Endereço da variável 'valor'.................: 0x7fffffffd658
Conteúdo do ponteiro x (endereço armazenado).: 0x7fffffffd658
Endereço da variável 'x'.....................: 0x7fffffffd660
Valor apontado por x (conteúdo de *x).........: 35
Conteúdo da variável y.......................: 35
```

Ponteiros

INSTITUTO FEDERAL
Pernambuco

- Um ponteiro permite alterar o valor de uma variável acessando diretamente o endereço onde ela está armazenada.

```c
int a = 40;

printf("O conteúdo armazenado em a é: %d \n", a);

int *pointer = &a;
*pointer = 30;

printf("O conteúdo armazenado em a é: %d \n", a);
```

Ponteiros

INSTITUTO FEDERAL
Pernambuco

- Um ponteiro permite alterar o valor de uma variável acessando diretamente o endereço onde ela está armazenada.

```c
int a = 40;

printf("O conteúdo armazenado em a é: %d \n", a);

int *pointer = &a;
*pointer = 30;

printf("O conteúdo armazenado em a é: %d \n", a);
```

```
O conteúdo armazenado em a é: 40
O conteúdo armazenado em a é: 30
```

Ponteiros

INSTITUTO FEDERAL
Pernambuco

- É possível atribuir um ponteiro a outro da mesma forma que fazemos com variáveis comuns.

```
int a = 30;
int *pointer1, *pointer2;
pointer1 = &a;
pointer2 = pointer1;

printf("O conteúdo armazenado em a é.....................: %d \n", a);
printf("O endereço de a é.....................: %p \n", &a);
printf("Conteúdo de pointer1 (endereço armazenado).....: %p\n", pointer1);
printf("Conteúdo de pointer2 (endereço armazenado).....: %p\n", pointer2);
printf("Valor apontado por pointer1 (conteúdo de *pointer1).: %d\n", *pointer1);
printf("Valor apontado por pointer2 (conteúdo de *pointer2).: %d\n", *pointer2);
```

Ponteiros

INSTITUTO FEDERAL
Pernambuco

```c
int a = 30;
int *pointer1, *pointer2;
pointer1 = &a;
pointer2 = pointer1;

printf("O conteúdo armazenado em a é.....................: %d \n", a);
printf("O endereço de a é................................: %p \n", &a);
printf("Conteúdo de pointer1 (endereço armazenado).......: %p\n", pointer1);
printf("Conteúdo de pointer2 (endereço armazenado).......: %p\n", pointer2);
printf("Valor apontado por pointer1 (conteúdo de *pointer1).: %d\n", *pointer1);
printf("Valor apontado por pointer2 (conteúdo de *pointer2).: %d\n", *pointer2);
```

```
O conteúdo armazenado em a é.....................: 30
O endereço de a é................................: 0x7fffffffd654
Conteúdo de pointer1 (endereço armazenado).......: 0x7fffffffd654
Conteúdo de pointer2 (endereço armazenado).......: 0x7fffffffd654
Valor apontado por pointer1 (conteúdo de *pointer1).: 30
Valor apontado por pointer2 (conteúdo de *pointer2).: 30
```

Ponteiros

INSTITUTO FEDERAL
Pernambuco

- A aritmética de ponteiros permite mover ponteiros pela memória.
- Ela é muito usada com:
  - vetores e matrizes;
  - strings;
  - alocação dinâmica e estruturas de dados.
- Exemplo:

```
int *p = &a;
p++;
```

- Ao executar p++, o ponteiro avança para o próximo endereço que pode armazenar um int. Isso equivale a avançar sizeof(tipo) bytes na memória.

Ponteiros

INSTITUTO FEDERAL
Pernambuco

- Em C, strings possuem uma relação muito forte com ponteiros.
- Isso acontece porque uma string é armazenada como uma sequência contínua de caracteres na memória.
- Em muitas expressões, o nome do vetor representa o endereço do primeiro elemento.

```
char nome[] = "IFPE";
printf("%c\n", *nome);
```

```
I
```

Ponteiros

INSTITUTO FEDERAL
Pernambuco

- Em C, strings possuem uma relação muito forte com ponteiros.
- Isso acontece porque uma string é armazenada como uma sequência contínua de caracteres na memória.
- Em muitas expressões, o nome do vetor representa o endereço do primeiro elemento.

```
char nome[] = "IFPE";
printf("%c\n", *nome);
```

nome -> endereço do primeiro caractere
*nome -> valor do primeiro caractere

Ponteiros

INSTITUTO FEDERAL
Pernambuco

- Equivalência:

```
nome[i]
```

```
*(nome + i)
```

- Exemplos:

```
char nome[] = "IFPE";
printf("%c\n", *nome);
printf("%c\n", *(nome + 1));
printf("%c\n", *(nome + 2));
printf("%c\n", *(nome + 3));
```

```
I
F
P
E
```

```
for (int i=0; i < (int)sizeof(nome)/sizeof(char) - 1; i++){
    printf("%c\n", *(nome + i));

}
```

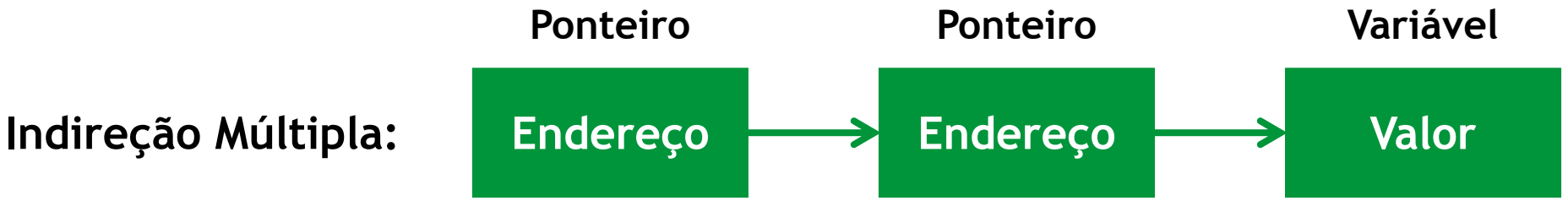
Ponteiros

INSTITUTO FEDERAL
Pernambuco

- É possível ter um ponteiro que aponte para outro ponteiro, o qual, por sua vez, aponta para uma variável comum armazenada na memória.
- Essa técnica é conhecida como indireção múltipla ou ponteiros para ponteiros, em contraste com a indireção simples, que ocorre quando um ponteiro aponta diretamente para uma variável.

Ponteiros

- É possível ter um ponteiro que aponte para outro ponteiro, o qual, por sua vez, aponta para uma variável comum armazenada na memória.
- Essa técnica é conhecida como indireção múltipla ou ponteiros para ponteiros, em contraste com a indireção simples, que ocorre quando um ponteiro aponta diretamente para uma variável.



Ponteiros

INSTITUTO FEDERAL
Pernambuco

- Um ponteiro para ponteiro deve ser declarado de forma específica. Para isso, adiciona-se um segundo caractere * na declaração da variável.
- Exemplo:

```
int **valor;
```

- Essa declaração cria um ponteiro chamado valor, que irá apontar para outro ponteiro, o qual, por sua vez, aponta para uma variável do tipo int.

Ponteiros

INSTITUTO FEDERAL
Pernambuco

- Um ponteiro para ponteiro deve ser declarado de forma específica. Para isso, adiciona-se um segundo caractere * na declaração da variável.
- Exemplo:

```
int **valor;
```

- Essa declaração cria um ponteiro chamado valor, que irá apontar para outro ponteiro, o qual, por sua vez, aponta para uma variável do tipo int.

Atenção: neste exemplo, valor NÃO é um ponteiro para um número inteiro, e sim um ponteiro para um ponteiro de inteiro.

Ponteiros

INSTITUTO FEDERAL
Pernambuco

```c
int a = 30;
int *pointer1, **pointer2;
pointer1 = &a;
pointer2 = &pointer1;

printf("Endereço de a.....................................: %p \n", &a);
printf("Conteúdo de pointer1 (endereço armazenado).....: %p\n", pointer1);
printf("Endereço de pointer1..............................: %p\n", &pointer1);
printf("Conteúdo de pointer2 (endereço armazenado).....: %p\n", pointer2);
printf("Valor apontado por pointer1 (conteúdo de *pointer1).: %d\n", *pointer1);
printf("Valor apontado por pointer2 (conteúdo de *pointer2).: %p\n", *pointer2);
printf("O valor armazenado em a é.........................: %d\n", **pointer2);
```

Ponteiros

INSTITUTO FEDERAL
Pernambuco

```c
int a = 30;
int *pointer1, **pointer2;
pointer1 = &a;
pointer2 = &pointer1;

printf("Endereço de a.....................................: %p \n", &a);
printf("Conteúdo de pointer1 (endereço armazenado).....: %p\n", pointer1);
printf("Endereço de pointer1..............................: %p\n", &pointer1);
printf("Conteúdo de pointer2 (endereço armazenado).....: %p\n", pointer2);
printf("Valor apontado por pointer1 (conteúdo de *pointer1).: %d\n", *pointer1);
printf("Valor apontado por pointer2 (conteúdo de *pointer2).: %p\n", *pointer2);
printf("O valor armazenado em a é.........................: %d\n", **pointer2);
```

```
Endereço de a.....................................: 0x7fffffffd654
Conteúdo de pointer1 (endereço armazenado).....: 0x7fffffffd654
Endereço de pointer1..............................: 0x7fffffffd658
Conteúdo de pointer2 (endereço armazenado).....: 0x7fffffffd658
Valor apontado por pointer1 (conteúdo de *pointer1).: 30
Valor apontado por pointer2 (conteúdo de *pointer2).: 0x7fffffffd654
O valor armazenado em a é.........................: 30
```

Exercícios

INSTITUTO FEDERAL
Pernambuco

1. Escreva um programa que declare um inteiro, um double e um char, e ponteiros para inteiro, double, e char. Associe as variáveis aos ponteiros (use &). Modifique os valores de cada variável usando os ponteiros. Imprima os valores das variáveis antes e após a modificação.
2. Escreva um programa que contenha duas variáveis inteiras. Leia essas variáveis do teclado. Em seguida, compare seus endereços e exiba o conteudo do maior endereço.

Exercícios

INSTITUTO FEDERAL
Pernambuco

3. O programa abaixo possui erro(s). Qual(is)? Como corrigí-lo(s)?

```c
#include <stdio.h>

int main(){

    int x, *p;
    x = 100;
    p = x;
    printf("Valor de p: %d. \n", *p);

    return 0;
}
```

Exercícios

INSTITUTO FEDERAL
Pernambuco

4. Implemente uma função:

```
void swap(int *x, int *y);
```

- A função deve trocar os valores de duas variáveis;
- Na main, leia dois inteiros do usuário;
- Chame swap(&a, &b) e mostre os valores antes e depois da troca.

Exercícios

INSTITUTO FEDERAL
Pernambuco

5. Qual o resultado do código abaixo? Explique cada linha.

```c
#include <stdio.h>

int main(){

    int x = 100, *p, **pp;
    p = &x;
    pp = &p;
    printf("Valor de *pp: %d. \n", **pp);

    return 0;
}
```

Exercícios

INSTITUTO FEDERAL
Pernambuco

6. Quais os valores de x e y ao final do trecho de código abaixo?

```c
#include <stdio.h>

int main(){

    int x, y, *p;
    y = 0;
    p = &y;
    x = *p;
    x = 4;
    (*p)++;
    --x;
    (*p) += x;
    printf("Valor de x: %d. \n", x);
    printf("Valor de y: %d. \n", y);

    return 0;
}
```

Exercícios

INSTITUTO FEDERAL
Pernambuco

6. Quais os valores de x e y ao final do trecho de código abaixo?

Atenção: o operador * tem menor precedência que os operadores ++ e +=.

Sem parênteses, o compilador incrementa o endereço armazenado no ponteiro, não o valor armazenado na variável apontada.

```c
#include <stdio.h>

int main(){

    int x, y, *p;
    y = 0;
    p = &y;
    x = *p;
    x = 4;
    (*p)++;
    --x;
    (*p) += x;
    printf("Valor de x: %d. \n", x);
    printf("Valor de y: %d. \n", y);

    return 0;
}
```

ALGORITMOS E ESTRUTURA DE DADOS

Curso de Engenharia de Software
Lucas Sampaio Leite