

Estruturas de Dados

Prof. Rodrigo Martins

rodrigo.martins@francomontoro.com.br

Cronograma da Aula

- Structs
- Ponteiros
- Ponteiros de Structs

Estrutura de Dados - STRUCT

- O conceito de orientação a objeto tem uma base muito sólida no conceito de estrutura de dados.
- Os structs em C++ são uma forma de definir tipos de dados personalizados que contêm um conjunto de campos (variáveis) que podem ter diferentes tipos de dados.
- As estruturas de dados consistem em criar apenas um dado que contém vários membros, que nada mais são do que outras variáveis.
- De uma forma mais simples, é como se uma variável tivesse outras variáveis dentro dela.
- A vantagem em se usar estruturas de dados é que podemos agrupar de forma organizada vários tipos de dados diferentes, por exemplo, dentro de uma estrutura de dados podemos ter juntos tanto um tipo float, um inteiro, um char ou um double.
- As variáveis que ficam dentro da estrutura de dados são chamadas de membros.

Criando uma estrutura de dados com STRUCT

- Para criar uma estrutura de dados usamos a palavra reservada struct.
- Toda estrutura deve ser criada antes de qualquer função ou mesmo da função principal main. Toda estrutura tem nome e seus membros são declarados dentro de um bloco de dados.
- Sintaxe:

```
struct nome_do_struct {  
    tipo_de_dado campo1;  
    tipo_de_dado campo2;  
    //...  
    tipo_de_dado campoN;  
};
```

Criando uma estrutura de dados com STRUCT

- Onde:
 - **nome_do_struct** é o nome que você dá ao struct.
 - **tipo_de_dado** é o tipo de dado que será armazenado em cada campo do struct.
 - **campo1, campo2, ..., campoN** são os nomes dos campos do struct.

Criando uma estrutura de dados com STRUCT

- Por exemplo, um struct que define um ponto no plano cartesiano pode ser definido da seguinte forma:

```
struct Ponto {  
    double x;  
    double y;  
};
```

- Para utilizar um struct em C++, basta criar uma variável do tipo do struct e acessar os seus campos utilizando o operador '.' Por exemplo:

```
Ponto p;  
p.x = 1.0;  
p.y = 2.0;
```

Criando uma estrutura de dados com STRUCT

- Os structs em C++ são uma forma útil de criar tipos de dados personalizados para armazenar informações estruturadas.
- Eles são frequentemente utilizados em programas que lidam com gráficos, jogos, bancos de dados e outros tipos de aplicações que envolvem dados estruturados.

struct1.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3
4  using namespace std;
5
6
7  struct data{
8      int dia;
9      int mes;
10     int ano;
11 };
12
13
14 int main (void){
15     data hoje;
16     hoje.dia = 24;
17     hoje.mes = 8;
18     hoje.ano = 2019;
19     cout <<"Hoje eh " << hoje.dia << "/" << hoje.mes << "/" << hoje.ano << endl;
20     system("pause");
21
22     return 0;
23 }
```

C:\Users\rodri\Google Drive\Uniesi\Estrutura de Dados\exemplos_Módulo

Hoje eh 24/8/2019

Pressione qualquer tecla para continuar. . .

Entendendo o Código

- A variável hoje é declarada como sendo um tipo de dado data. Data é uma estrutura de dados que tem três características (ou três membros) inteiros: dia, mes e ano.
- Como hoje é um tipo de dado data, ele obtém os mesmos três membros.
- Para acessar cada membro, usamos a variável e depois o nome do membro que queremos acessar separados por ponto (.).

struct1.1.cpp

```
*strcut1.1.cpp
1  #include <iostream>
2  #include <stdio.h>
3  #include <string>
4
5  using namespace std;
6
7  struct pessoa
8  {
9      char nome[100];
10     int idade;
11 };
12
13
14 int main()
15 {
16     pessoa p[100];
17     cout << "Qual seu Nome: " << endl;
18     //cin.ignore();
19     cin.getline(p[0].nome, sizeof(p[0].nome));
20     cout << "Quantos anos voce tem? ";
21     cin >> p[0].idade;
22     cout << "Idade: " << p[0].idade << " anos" << endl;
23     cout << "Nome: " << p[0].nome << endl;
24
25     return 0;
26 }
```

Typedef

- Em C++ podemos redefinir um tipo de dado dando-lhe um novo nome.
- Essa forma de programação ajuda em dois sentidos:
 - 1º. Fica mais simples entender para que serve tal tipo de dado;
 - 2º. É a única forma de conseguirmos referenciar uma estrutura de dados dentro de outra (struct dentro de struct).

Typedef

- Typedef deve sempre vir antes de qualquer programação que envolva procedimentos (protótipo de funções, funções, função main, structs, etc.).
- Sua sintaxe base é:

```
typedef tipo_de_dado novo_nome;
```

- Onde:
 - **tipo_de_dado** é o tipo de dado que você deseja criar um novo nome.
 - **novo_nome** é o novo nome que você deseja criar para o tipo de dado.

Typedef

- Por exemplo, você pode criar um novo nome Meulnt para o tipo de dado int da seguinte forma:

```
typedef int Meulnt;
```

- Agora, sempre que você utilizar o nome Meulnt no código, ele será interpretado como o tipo de dado int. Por exemplo:

```
Meulnt x = 42;  
cout << "x = " << x << endl;
```

Typedef

- O typedef pode ser especialmente útil quando você está lidando com tipos de dados complexos e longos, como ponteiros para funções ou estruturas com muitos campos.
- Com o typedef, você pode criar nomes mais curtos e legíveis para esses tipos de dados complexos, tornando o código mais fácil de ler e entender.

typedef1.cpp

```
typedef1.cpp x
1  #include <iostream>
2  #include <cstdlib>
3
4  using namespace std;
5
6
7  typedef int inteiro;
8  typedef float decimal;
9
10
11
12  int main () {
13      inteiro x = 1;
14      decimal y = 1.5;
15      cout << "X=" << x << endl << "Y=" << y << endl;
16
17      system ("pause");
18      return 0;
19  }
```

C:\Users\rodri\Google Drive\Uniesi\Estrutura de Dados\exemplos_Mc

X=1
Y=1.5
Pressione qualquer tecla para continuar. . .

Definindo nomes para estruturas de dados

- Uma vantagem muito grande que typedef nos proporciona é definir um nome para nossa estrutura de dados (struct).
- Graças a isso, somos capazes de auto-referenciar a estrutura, ou seja, colocar um tipo de dado struct dentro de outro struct.
- Podemos definir o nome de uma estrutura de dados (struct) de duas maneiras:
 - Definindo o nome da estrutura e só depois definir a estrutura; ou
 - definir a estrutura ao mesmo tempo que define o nome.

Definindo nomes para estruturas de dados

- Da primeira forma:

```
typedef struct estrutura1 MinhaEstrutura;
```

```
    struct estrutura1 {  
        int var1;  
        float var2;  
    };
```

- Da segunda forma:

```
typedef struct estrutura1 {  
    int var1;  
    float var2;  
} MinhaEstrutura;
```

typedef2.cpp

*typedef2.cpp

x

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <cstring>
4
5  using namespace std;
6
7  typedef struct data {
8      //para declarar que um número seja apenas positivo (incluindo o 0),
9      //usamos o modificador unsigned:
10     unsigned short dia;
11     unsigned short mes;
12     unsigned int ano;
13 } Data;
14
15 typedef struct aniversario {
16     char nome[50];
17     Data nascimento;
18 } Aniversario;
19
```

typedef2.cpp

```
20 int main (){
21     Aniversario alguem;
22     cout << "Digite o nome de alguem" << endl;
23     cin >> alguem.nome;
24     cout << "Digite o dia que esta pessoa nasceu" << endl;
25     cin >> alguem.nascimento.dia;
26     cout << "Digite o mes que esta pessoa nasceu" << endl;
27     cin >> alguem.nascimento.mes;
28     cout << "Digite o ano que esta pessoa nasceu" << endl;
29     cin >> alguem.nascimento.ano;
30     system ("cls");
31
32     cout << alguem.nome << endl;
33     cout << "nasceu em ";
34     cout << alguem.nascimento.dia<<"/";
35     cout << alguem.nascimento.mes<<"/";
36     cout << alguem.nascimento.ano<<endl;
37
38     system ("pause");
39     return 0;
40 }
```

Exercícios

1. Crie uma estrutura chamada pessoa que seja capaz de armazenar o nome, o endereço, o CPF e a idade de 5 pessoas.

Exercícios

2. Suponha que você está desenvolvendo um programa para armazenar informações sobre animais em um zoológico. Crie uma estrutura chamada `Animal` que deve armazenar as seguintes informações sobre cada animal:

- Nome do animal (string)
- Espécie do animal (string)
- Idade do animal (int)
- Peso do animal em kg (float)

Em seguida, crie um `typedef` para uma lista de animais chamada `ListaAnimais`, que deve ser um vetor de 10 elementos do tipo `Animal`.

Por fim, crie um programa que pede ao usuário para digitar os dados de 10 animais e armazena esses dados em uma variável do tipo `ListaAnimais`.

Ao final, o programa deve exibir os dados de cada um dos 10 animais na tela.

Ponteiros

- A memória RAM de um computador é um conjunto de posições adjacentes.
- De uma maneira simplista podemos dizer que ela é um grande vetor e seu índice é formado pelos endereços individuais de memória.
- Os ponteiros em C++ são uma das características mais importantes da linguagem e permitem manipular diretamente a memória do computador.
- Com ponteiros, é possível criar estruturas de dados complexas e executar operações de baixo nível que não seriam possíveis com outros tipos de variáveis.

Ponteiros

- Em C++, um ponteiro é uma variável que armazena o endereço de memória de outra variável.
- Um ponteiro é declarado usando o operador de asterisco *, que é colocado antes do nome da variável. Por exemplo:

```
int* ptr;
```

- Nesse exemplo, ptr é um ponteiro para um valor inteiro.
- Ele não armazena o valor inteiro em si, mas sim o endereço de memória onde o valor está armazenado.

Ponteiros

- Para obter o endereço de memória de uma variável, use o operador de endereço &, que é colocado antes do nome da variável. Por exemplo:

```
int x = 42;  
int* ptr = &x;
```

- Nesse exemplo, &x retorna o endereço de memória da variável x e o valor desse endereço é armazenado no ponteiro ptr.

Ponteiros

- Para acessar o valor armazenado em um endereço de memória apontado por um ponteiro, use o operador de referência *, que é colocado antes do nome do ponteiro. Por exemplo:

```
int y = *ptr;
```

- Nesse exemplo, *ptr retorna o valor armazenado no endereço de memória apontado pelo ponteiro ptr e esse valor é armazenado na variável y.

Ponteiros

- Os ponteiros também podem ser utilizados para criar e manipular estruturas de dados complexas, como listas encadeadas, árvores binárias, grafos, entre outras.
- No entanto, é importante tomar cuidado ao manipular ponteiros, pois eles podem facilmente causar erros de segmentação (também conhecidos como "segfaults"), que ocorrem quando um programa tenta acessar uma área inválida da memória.

Como uma variável é armazenada?

- Imagine que temos este comando em C++:

`char ch;`

- O compilador determina um endereço de memória para a variável.

Endereço	Variável	Conteúdo
1000		
1001		
1002	ch	
1003		
1004		
1005		

Como uma variável é armazenada?

- Quando fazemos:

`ch = 'A';`

- estamos dizendo ao compilador para colocar o 'A' na posição de memória de ch.

Endereço	Variável	Conteúdo
1000		
1001		
1002	ch	A
1003		
1004		
1005		

Como uma variável é armazenada?

- Imagine agora que nós criássemos um ponteiro ptr.
- Como qualquer outra variável ele ocuparia espaço.

Endereço	Variável	Conteúdo
1000	ptr	
1001		
1002	ch	A
1003		
1004		
1005		

Como uma variável é armazenada?

- Se rodássemos o seguinte comando:

`ptr = &ch;`

- Temos o seguinte resultado e podemos dizer que ptr aponta para ch.

Endereço	Variável	Conteúdo
1000	ptr	1002
1001		
1002	ch	A
1003		
1004		
1005		



Declaração de ponteiros

- Devemos ter em mente que um ponteiro é uma variável como outra qualquer, e, por isso, deve ser declarado.

- Para declarar um ponteiro usamos:

`tipo *ponteiro;`

- Ex

`char a, b, *ptr , c, *x;`

`int v1, *p1;`

Inicialização de ponteiros

- Todo ponteiro deve ser inicializado, como qualquer variável.
- Um ponteiro zerado nunca pode ser usado, mas podemos inicializa-lo.
- Existe uma constante em C++ que você pode utilizar para inicializar um ponteiro: NULL
- Ela nada mais é que um “apelido” para o número zero.

Inicialização de ponteiros

- Vejamos o código:

```
int a=5, b=7;
```

```
int *ptr = NULL;
```

Endereço	Variável	Conteúdo
1000	ptr	NULL
1001		
1002	a	5
1003	b	7
1004		
1005		


Inicialização de ponteiros

- Se agora usarmos:

`ptr = &a;`

- Agora temos: $a \rightarrow 5$ $ptr \rightarrow 1002$ $*ptr \rightarrow 5$

Endereço	Variável	Conteúdo
1000	ptr	1002
1001		
1002	a	5
1003	b	7
1004		
1005		



exemplo1.cpp

```
exemplo1.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char** argv)
6  {
7      int var = 10; // declaração de variável padrão
8      int *pvar; // declaração de ponteiro
9      //sempre inicializar o ponteiro antes de utiliza-lo
10     pvar = &var; // &var --> leia como o endereço da variável var
11     *pvar = 20; // *pvar muda o valor de var, porque o ponteiro tem o seu endereço
12     cout << *pvar << endl; // *pvar mostra o conteúdo da variável apontada pelo ponteiro
13     cout << var << endl;
14     cout << &*pvar << endl;
15     return 0;
16 }
```

```
C:\Users\rodri\Google Drive\Uniesi\Estrutura de Dado...
20
20
0x67fef8

O Processo retornou 0   tempo de execução : 0.077 s
Pressione uma tecla para continuar...
```

exemplo1.1.cpp

exemplo1.1.cpp

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char** argv)
6  {
7      short usVar = 200;
8      long ulVar = 300;
9      int iVar = 400;
10     cout << "*** Valores e enderecos ***" << endl;
11     cout << "usVar: Valor = " << usVar << ", Endereco = " << &usVar << endl;
12     cout << "ulVar: Valor = " << ulVar << ", Endereco = " << &ulVar << endl;
13     cout << "iVar: Valor = " << iVar << ", Endereco = " << &iVar << endl;
14     return 0;
15 }
```

C:\Users\rodri\Google Drive\Uniesi\Estrutura de Dados\exemplos_Módulo_3\

```
*** Valores e enderecos ***
usVar: Valor = 200, Endereco = 0x67fee4
ulVar: Valor = 300, Endereco = 0x67fee8
iVar: Valor = 400, Endereco = 0x67fee4
```

```
O Processo retornou 0   tempo de execução : 0.124 s
Pressione uma tecla para continuar...
```

exemplo2.cpp

```
exemplo2.cpp x
1  #include <iostream>
2
3  using namespace std;
4
5  //escopo global
6  int var = 0;
7
8  void passagemPorValor(int var){
9      var = 20;
10 }
11
12 int main(int argc, char** argv)
13 {
14     var = 10;
15     int *pvar;
16
17     pvar = &var;
18
19     passagemPorValor(var);
20
21     cout << var << endl;
22     return 0;
23 }
```

10

0 Processo retornou 0 tempo de execução : 0.061 s
Pressione uma tecla para continuar...

exemplo3.cpp

```
exemplo3.cpp x
1  #include <iostream>
2
3  using namespace std;
4
5
6  int var = 0;
7
8  void passagemPorReferencia(int* n){ // *n aponta para o endereço de pvar
9      *n = 20;
10 }
11
12 int main(int argc, char** argv)
13 {
14     var = 10;
15
16     int *pvar;
17     pvar = &var;
18
19     passagemPorReferencia(pvar);
20
21     cout << var << endl;
22     return 0;
23 }
```

20

O Processo retornou 0 tempo de execução : 0.085 s
Pressione uma tecla para continuar...

exemplo4.cpp

```
*exemplo4.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  //exemplo de ponteiro de array
6
7  int main(int argc, char** argv)
8  {
9      int array[] = {1,2,3,4,5};
10     int* pArray = &array[0];
11
12     cout << *pArray << endl;
13     cout << endl;
14
15     for(int i=0;i<5;i++){
16         cout << *pArray << endl;
17         pArray++;
18     }
19
20     return 0;
21 }
```

C:\Users\roam\Google Drive\Onlres\Estrutura de Da...

1
1
2
3
4
5

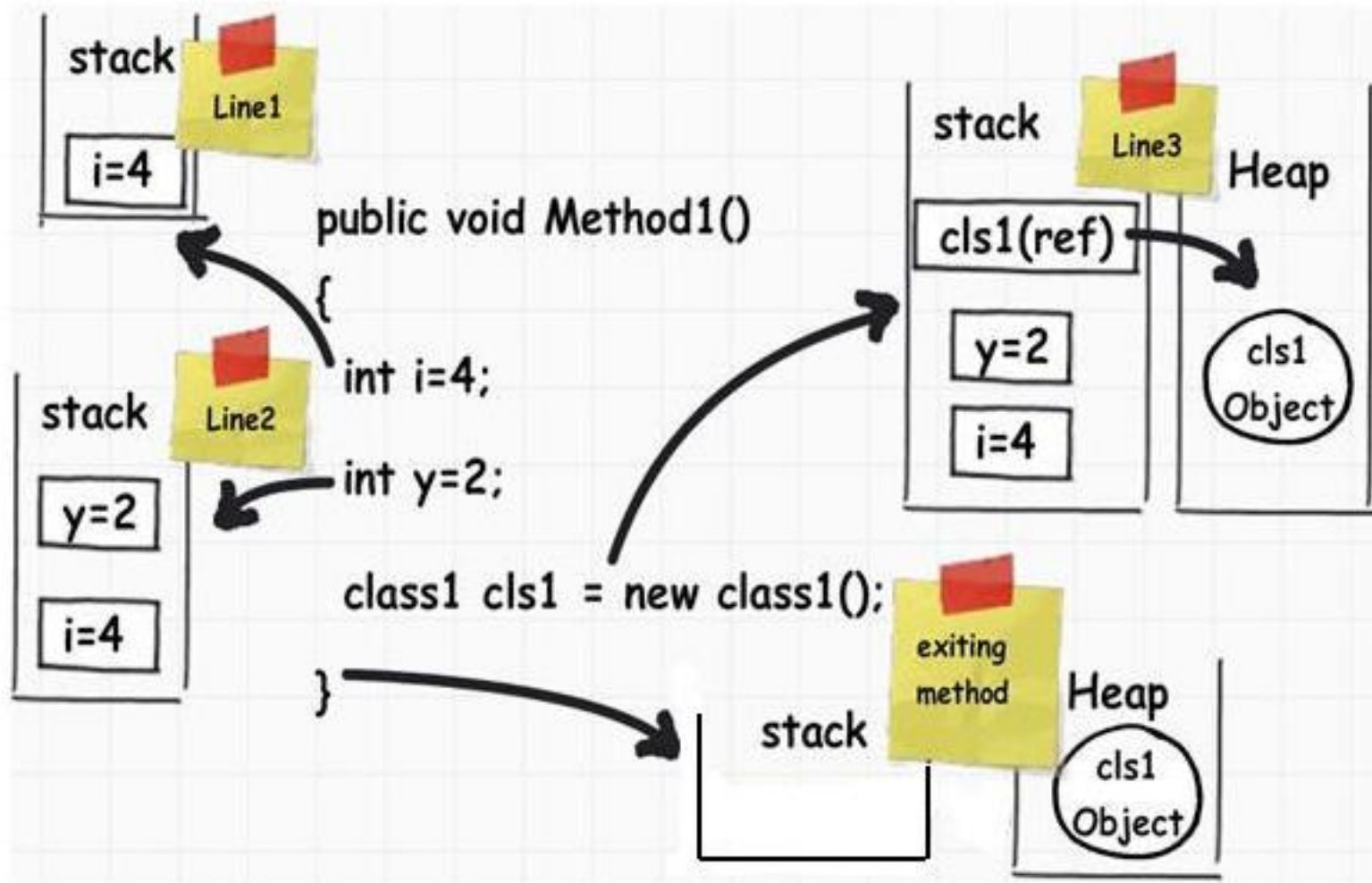
O Processo retornou 0 tempo de execução : 0.094 s
Pressione uma tecla para continuar...

exemplo4.1.cpp

```
*exemplo4.1.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  //exemplo de alocação dinâmica de memória
6
7
8  int main()
9  {
10     int *ptr_a;
11
12     // new ---> cria a área necessária para 01 inteiro e
13     // coloca em 'ptr_a' o endereço desta área.
14     ptr_a = new int;
15
16     cout << "Endereco de ptr_a: " << ptr_a << endl;
17     *ptr_a = 90;
18     cout << "Conteudo de ptr_a: " << *ptr_a << endl;
19
20     // Quando o valor já não é necessário pode ser liberada a memória
21     // reservada através do comando delete.
22     delete ptr_a;
23 }
```

```
Endereco de ptr_a: 0xda2158
Conteudo de ptr_a: 90
```

```
O Processo retornou 0    tempo de execução : 0.475 s
Pressione uma tecla para continuar...
```

exemplo6.cpp

```
*exemplo6.cpp
1  #include <iostream>
2  #include <cstdlib>
3
4  using namespace std;
5
6  //Ponteiro de Struct
7
8  typedef struct data {
9      short dia;
10     short mes;
11     int ano;
12 } Data;
13
14 int main (void){
15     Data data; //variável data do tipo struct data
16     Data *hoje; //ponteiro hoje para um tipo struct data
17     hoje = &data; //hoje aponta para o endereço de data
18
19     //dados sendo inseridos na variável data
20     (*hoje).dia = 20;
21     (*hoje).mes = 1;
22     (*hoje).ano = 2009;
23
24     //hoje->dia = 20;
25     //hoje->mes = 1;
26     //hoje->ano = 2009;
27
28     //mostrando o que está gravado no endereço contido em hoje
29     cout << "Data registrada:" << endl;
30     cout << hoje->dia << "/" << hoje->mes << "/" << hoje->ano << endl;
31     system ("pause");
32 }
```

Exercícios

1. Indique verdadeiro ou falso
 - a) () O operador & permite-nos obter o endereço de uma variável. Permite também obter o endereço de um ponteiro.
 - b) () Se x é um inteiro e ptr um ponteiro para inteiros e ambos contêm no seu interior o número 100, então x+1 e ptr+1 apresentarão o número 101.
 - c) () O operador * nos permite obter o endereço de uma variável.
 - d) () Os ponteiros são variáveis que apontam para endereços na memória.

Exercícios

2. Qual o resultado?

```
cout << a << b << *ptr;
```

3. Se fizermos `ptr = &b`, qual o resultado?

```
cout << a << b << *ptr;
```

4. Se agora tivermos `*ptr = 20`, qual o resultado?

Endereço	Variável	Conteúdo
1000	ptr	1002
1001		
1002	a	5
1003	b	7
1004		

Exercícios

5. Qual caractere que se coloca na declaração de uma variável para indicar que ela é um ponteiro? Onde se coloca este caractere?
6. O que contém uma variável do tipo ponteiro?
7. Faça um programa em C++ que crie um vetor de 10 inteiros, coloque peça valores ao usuário e depois imprima todos os seus conteúdos na ordem normal e depois inversa. A impressão dos conteúdos deverá ser feita usando ponteiro.

Exercícios

8. Escreva um programa em C++ que crie um array de 5 inteiros, preencha-o com valores fornecidos pelo usuário e, em seguida, calcule a média dos valores usando ponteiros.

Dica: para calcular a média, some todos os valores do array e divida o resultado pelo número de elementos no array.

Referência desta aula

- Notas de Aula do Prof. Prof. Armando Luiz N. Delgado, baseado em revisão sobre material de Prof.a Carmem Hara e Prof. Wagner Zola.
- VELOSO, P. et Alli. Estruturas de Dados. Ed. Campus, 1986.
- <http://www.cplusplus.com/reference/>

Obrigado

Rodrigo