Estruturas de Dados

Prof. Rodrigo Martins rodrimartins2005@gmail.com



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.
* 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

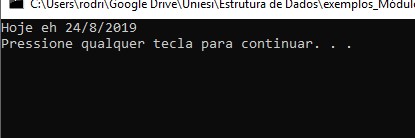
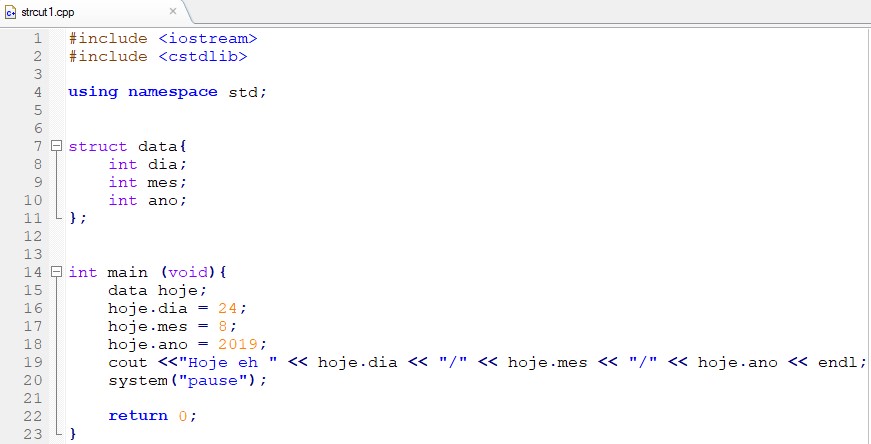
STRUCT

* 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\_da\_estrutura { tipo\_de\_dado nome\_do\_membro;

};

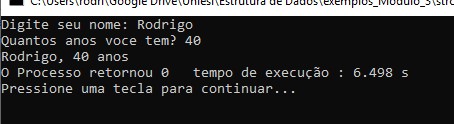
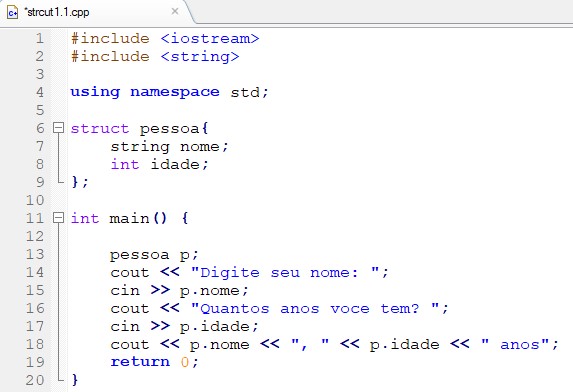
# struct1.cpp



# 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



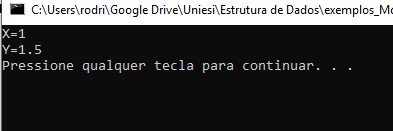
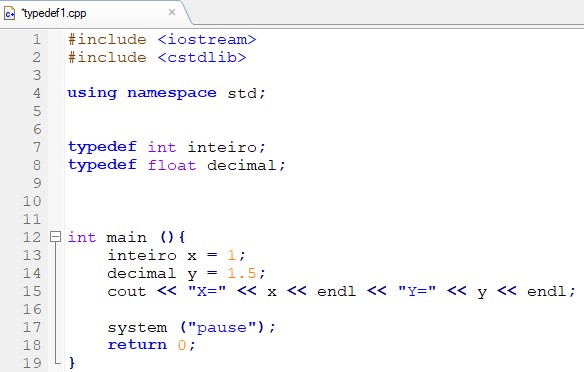
# Typedef

* 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.) e sua sintase base é: typedef nome\_antigo nome novo;

# typedef1.cpp



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;

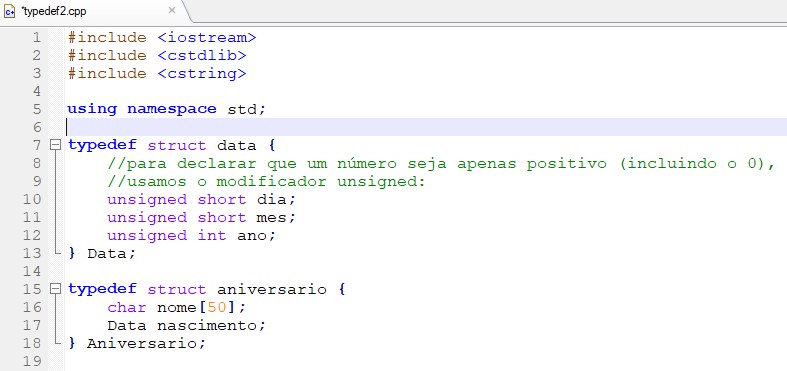
float var2;

};

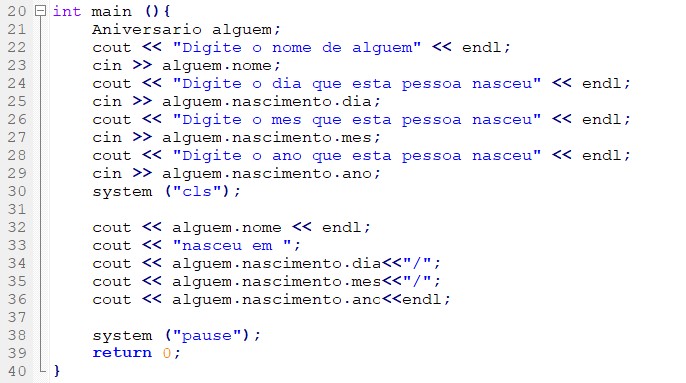
* Da segunda forma:

typedef struct estrutura1 { int var1; float var2;

} MinhaEstrutura;

typedef2.cpp

# typedef2.cpp



# Exercícios

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

# 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 nada mais são que este índice.
* Ponteiro é uma variável que contém o endereço de outra variável na memória interna do computador.
* Imagine que temos este comando em C++: char ch;

**Endereço**

**Variável**

**Conteúdo**

1000

1001

1002

ch

1003

1004

1005

* O compilador determina um endereço de memória para a variável.
* Quando fazemos:

|  |  |  |
| --- | --- | --- |
| **Endereço** | **Variável** | **Conteúdo** |
| 1000 |  |  |
| 1001 |  |  |
| 1002 | ch | A |
| 1003 |  |  |
| 1004 |  |  |
| 1005 |  |  |

ch = ‘A’;

* estamos dizendo ao compilador para colocar o ‘A’ na posição de memória de ch.

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

* Imagine agora que nós criássemos um ponteiro ptr.
* Como qualquer outra variável ele ocuparia espaço.
* Se rodássemos o seguinte comando: ptr = &ch;

**Endereço**

**Variável**

**Conteúdo**

1000

ptr

1002

1001

1002

ch

A

1003

1004

1005

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

# 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;
* 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.
* 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 |  |  |

* Se agora usarmos:

ptr = &a;

* Agora temos: a → 5 ptr → 1002 \*ptr → 5

**Endereço**

**Variável**

**Conteúdo**

1000

ptr

1002

1001

1002

a

5

1003

b

7

1004

1005

* Existem 4 operações possíveis com ponteiros:

–Incremento

–Decremento

–Diferença

–Comparação

* Incremento:

–Adiciona 1 tamanho do tipo da variável apontada no ponteiro, ou seja: ptr = ptr + sizeof(tipo)

* Decremento:

–Subtrai 1 tamanho do tipo da variável apontada no ponteiro, ou seja ptr = ptr - sizeof(tipo)

* Diferença:

–Determina quantos elementos, do tipo que os ponteiros apontam, existem entre os ponteiros:

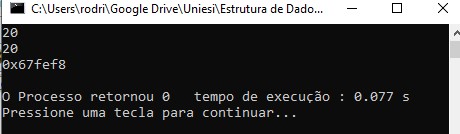
–A diferença de ponteiros não é tão utilizada.

* Comparação:

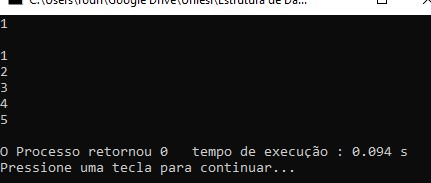
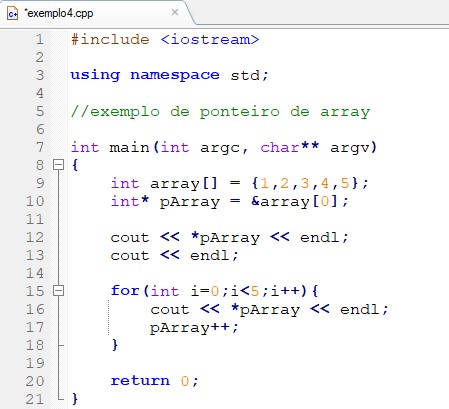
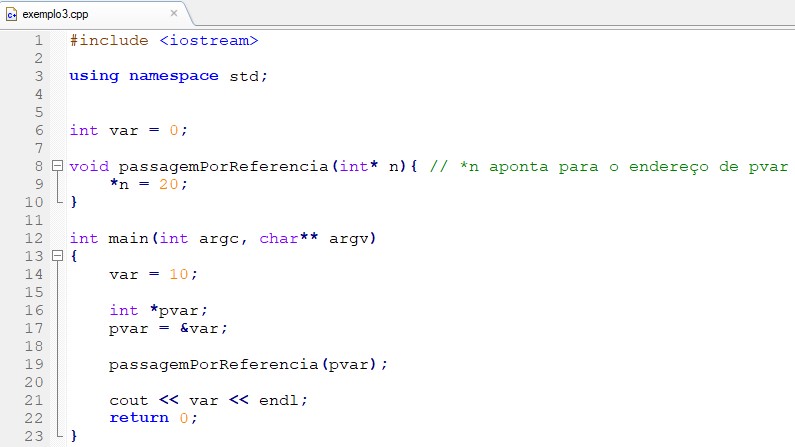
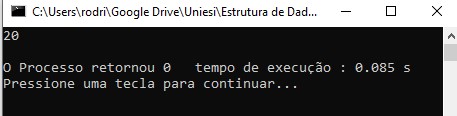
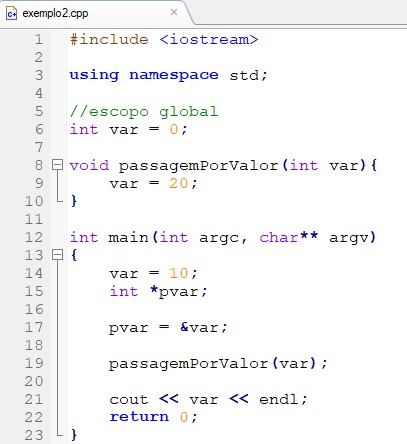
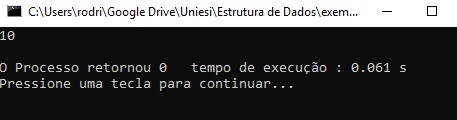
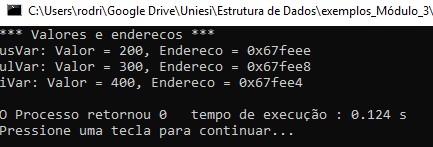
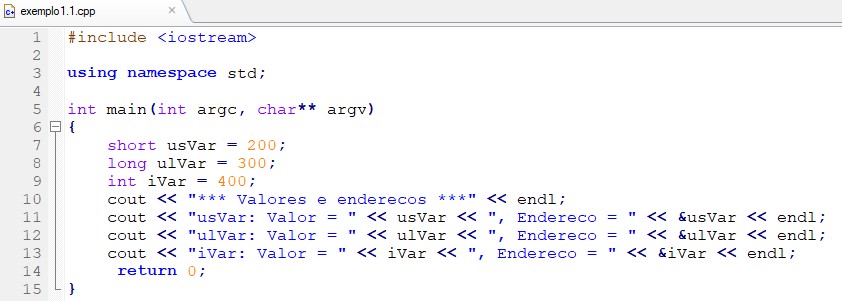
–Determina se um ponteiro vem antes na memória do que o outro.

–A comparação também não é tão usada.

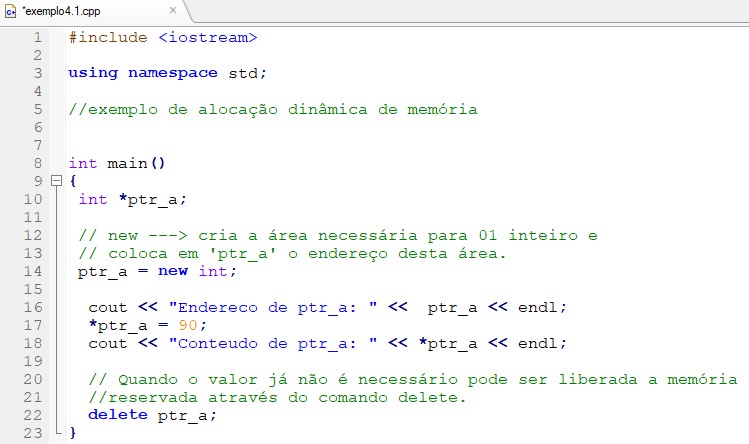
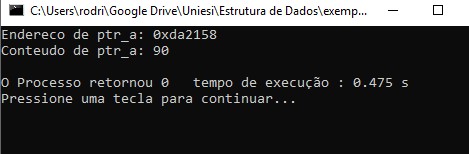
# exemplo1.cpp

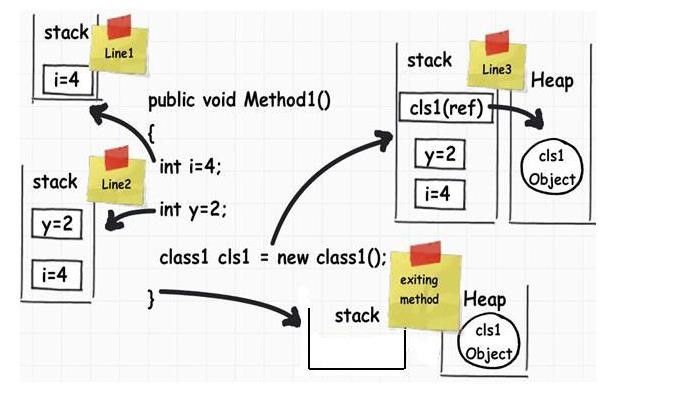


# exemplo1.1.cpp

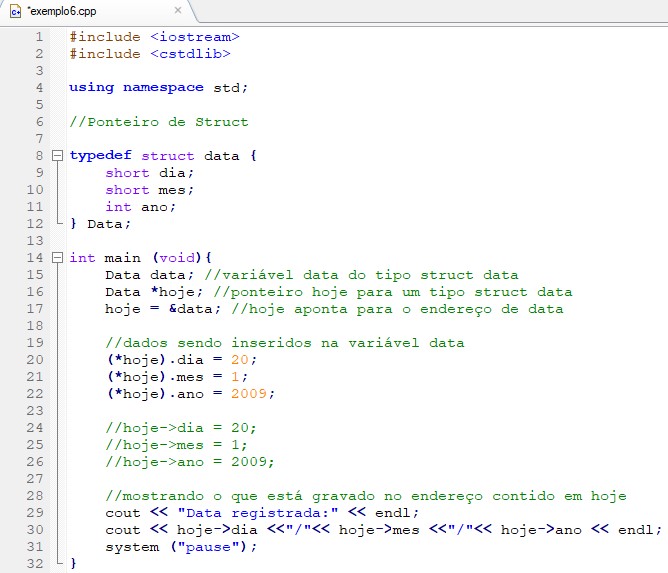


# exemplo4.1.cpp





# exemplo6.cpp



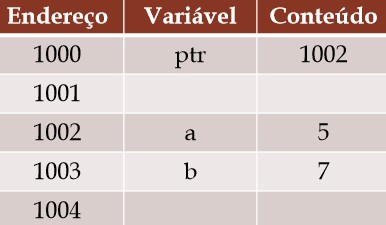
1. Indique verdadeiro ou falso

1. (V) O operador & permite-nos obter o endereço de uma variável. Permite também obter o endereço de um ponteiro.
2. (V) 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.
3. (F) O operador \* nos permite obter o endereço de uma variável.
4. (V) Os ponteiros são variáveis que apontam para endereços na memória.
5. Qual o resultado?

cout << a << b << \*ptr;

R: 5 7 5

1. Se fizermos ptr = &b, qual o resultado? cout << a << b << \*ptr; R: 5 7 7
2. Se agora tivermos \*ptr = 20, qual o resultado? R: 5 20 20



1. Qual caractere que se coloca na declaração de uma variável para indicar que ela é um ponteiro? Onde se coloca este caractere? R: \*(asterisco), antes do nome da variável que será o seu ponteiro, exemplo: tipo \*nomePonteiro.
2. O que contém uma variável do tipo ponteiro? R: Contém o endereço de memória de outra variável.
3. Faça um programa que crie um vetor de 10 inteiros, coloque valores nele e depois imprima todos os seus conteúdos na ordem normal e depois inversa. A impressão dos conteúdos deverá ser feita usando ponteiro.

# 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
* http://www.cplusplus.com/reference/

Obrigado

Rodrigo