Residência em Software

Aula 11
Tipos de dados estruturados

Professores:

Alvaro Degas Coelho,
Edgar Alexander,
Esbel Thomas Valero,
Helder Conceição Almeida

INSTITUIÇÃO EXECUTORA
cepedi UESC

COORDENADORA
MCTI FUTURO Softex

APOIO
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Residência em Software

Tipos de dados estruturados

- Uma estrutura (struct) é um tipo de dado que permite agrupar diferentes tipos de variáveis sob um único nome.
- É semelhante a uma classe, mas possui membros públicos por padrão (a partir do C++11, também pode ter membros privados e protegidos).

Residência em Software

Tipos de dados estruturados

- Uma estrutura é útil quando você deseja armazenar dados relacionados em uma única entidade.
- As variáveis pertencentes a uma estrutura são denominadas membros, elementos ou campos da estrutura.
- Os elementos da estrutura devem ser logicamente relacionados.

Residência em Software

Struct

- Para definir uma estrutura em C++, você usa a palavra-chave struct seguida pelo nome da estrutura e, opcionalmente, uma lista de membros dentro de chaves {}.
- Cada membro é uma variável com um tipo de dado associado.
- Os **membros** de uma estrutura **podem ser de qualquer tipo**, incluindo outros tipos de estrutura, ponteiros, arrays e assim por diante.

Residência em Software

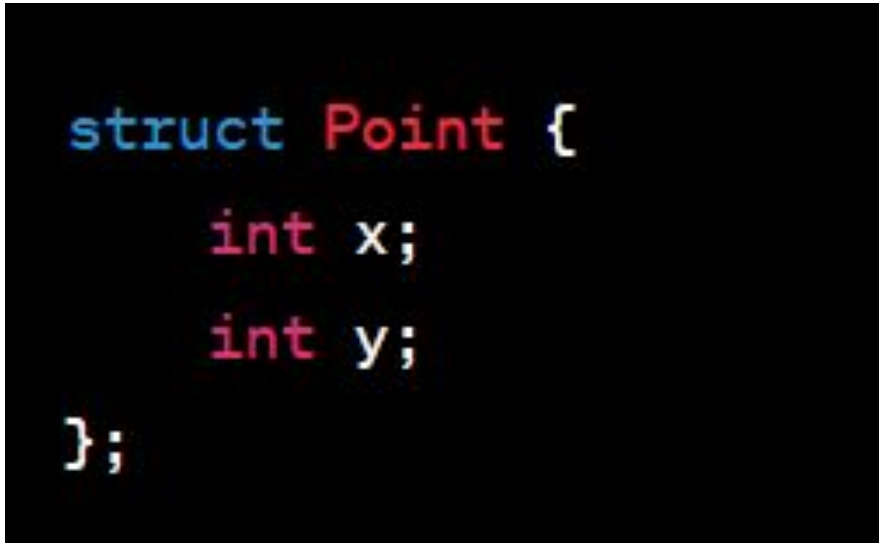
POD (Plain Old Data) Structure

- A **POD structure** é uma struct que **somente tem variáveis como membro**.

- Não contém funções virtuais, métodos, construtores, destrutores e etc…

Residência em Software

Exemplo simples de uma estrutura que representa um ponto no plano cartesiano



Neste exemplo, criamos uma estrutura chamada Point com dois membros inteiros x e y.

Residência em Software

Exemplo – Acessando os dados da struct

```
int main() {
    Point p1;       // Criando uma variável do tipo Point
    p1.x = 10;      // Atribuindo valor a 'x' da variável 'p1'
    p1.y = 20;      // Atribuindo valor a 'y' da variável 'p1'

    Point p2 = {5, 15};  // Inicializando a variável 'p2' com valores para

    // Acessando os membros 'x' e 'y' da variável 'p1'
    std::cout << "p1.x: " << p1.x << ", p1.y: " << p1.y << std::endl;

    // Acessando os membros 'x' e 'y' da variável 'p2'
    std::cout << "p2.x: " << p2.x << ", p2.y: " << p2.y << std::endl;

    return 0;
}
```

Residência em Software

Exemplos de POD Structs

```cpp
#include <iostream>
using namespace std;

struct PERSON {   // Declare PERSON struct type
    int age;   // Declare member types
    long ss;
    float weight;
    char name[25];
} family_member;   // Define object of type PERSON

struct CELL {   // Declare CELL bit field
    unsigned short character  : 8;  // 00000000 ????????
    unsigned short foreground : 3;  // 00000??? 00000000
    unsigned short intensity  : 1;  // 0000?000 00000000
    unsigned short background : 3;  // 0???0000 00000000
    unsigned short blink      : 1;  // ?0000000 00000000
} screen[25][80];       // Array of bit fields

int main() {
    struct PERSON sister;   // C style structure declaration
    PERSON brother;   // C++ style structure declaration
    sister.age = 13;   // assign values to members
    brother.age = 7;
    cout << "sister.age = " << sister.age << '\n';
    cout << "brother.age = " << brother.age << '\n';

    CELL my_cell;
    my_cell.character = 1;
    cout << "my_cell.character = " << my_cell.character;
}
// Output:
// sister.age = 13
// brother.age = 7
// my_cell.character = 1
```

Fonte: https://learn.microsoft.com/pt-br/cpp/cpp/struct-cpp?view=msvc-170

Residência em Software

Union

- Registro (union) é uma construção semelhante a uma estrutura, mas **todos os membros compartilham o mesmo espaço de memória**. Ou seja, um registro permite que você armazene diferentes tipos de dados, mas **apenas um dos membros pode ser usado em um determinado momento**.
- Isso pode ser útil quando você precisa de uma forma de economizar espaço em memória e precisa armazenar tipos diferentes, **mas nunca usá-los simultaneamente**.
- Para definir um registro em C++, você usa a palavra-chave union seguida pelo nome do registro e uma lista de membros dentro de chaves {}.

Residência em Software

Exemplo

```
union Value {
    int intValue;
    char charValue;
};
```

- No exemplo ao lado, criamos um registro chamado Value com dois membros intValue e charValue, representando um inteiro e um caractere, respectivamente.
- Ao usar um registro, é fundamental garantir que você esteja acessando o membro correto, uma vez que ambos compartilham o mesmo espaço de memória.

Residência em Software

Estruturas e Registros

- Em resumo, **estruturas** em C++ são utilizadas para agrupar diferentes tipos de variáveis sob um único nome, enquanto **registros** são usados para compartilhar o mesmo espaço de memória para diferentes tipos de dados.

- Cada um tem sua aplicação específica dependendo dos requisitos do programa.

Residência em Software

Definição de Tipos

- Em C++, a palavra-chave ***typedef*** é usada para criar sinônimos (ou alias) para tipos de dados existentes.
- Com ***typedef***, você pode criar um novo nome para um tipo de dado já definido, tornando o código mais legível e facilitando a manutenção, especialmente quando se trata de tipos de dados complexos ou longos.

Residência em Software

Sintaxe de typedef

- A sintaxe geral do typedef é a seguinte:

```
typedef tipo_existente novo_nome;
```

Residência em Software

Exemplo 1

```c
typedef int MeuInteiro; // Cria um alias "MeuInteiro" para o tipo int
 int main()
{
      MeuInteiro numero = 42; // Usando o novo nome para int
                                          // Agora, 'MeuInteiro' é trat
     return 0;
}
```

Residência em Software

Exemplo 2 – Alias para uma struct

```c
struct Ponto {
    int x;
    int y;
};

typedef Ponto MeuPonto;  // Cria um alias "MeuPonto" para a estrutura Ponto

int main() {
    MeuPonto p;   // Usando o novo nome para a estrutura Ponto
    p.x = 10;
    p.y = 20;

    // Agora, 'MeuPonto' é tratado como sinônimo de 'Ponto'.
    return 0;
}
```

Residência em Software

Exemplo 3 – Alias para um ponteiro de função

```c
typedef int (*OperacaoMatematica)(int, int);

int Soma(int a, int b) {
    return a + b;
}

int main() {
    OperacaoMatematica somaFunc = &Soma; // Cria um alias para um ponteiro p

    int resultado = somaFunc(5, 3); // Chama a função Soma usando o ponteiro
    // resultado = 8
    return 0;
}
```

Residência em Software

Typedef

- O uso do **typedef** pode ser particularmente útil em situações onde você está trabalhando com tipos de dados complexos, como estruturas aninhadas, ponteiros para funções e assim por diante.

- Ele ajuda a tornar o código mais legível, auto documentado e facilita possíveis alterações futuras nos tipos de dados utilizados em seu programa.

Residência em Software

Utilizando o *using* ao invés de *typedef*

- A partir do C++11, você pode usar a palavra-chave **using** para obter resultados semelhantes ao **typedef**, de uma forma mais legível:

```
using novo_nome = tipo_existente;
```

Exemplo:

```
using MeuInteiro = int; // Cria um alias "MeuInteiro" para o tipo int
```